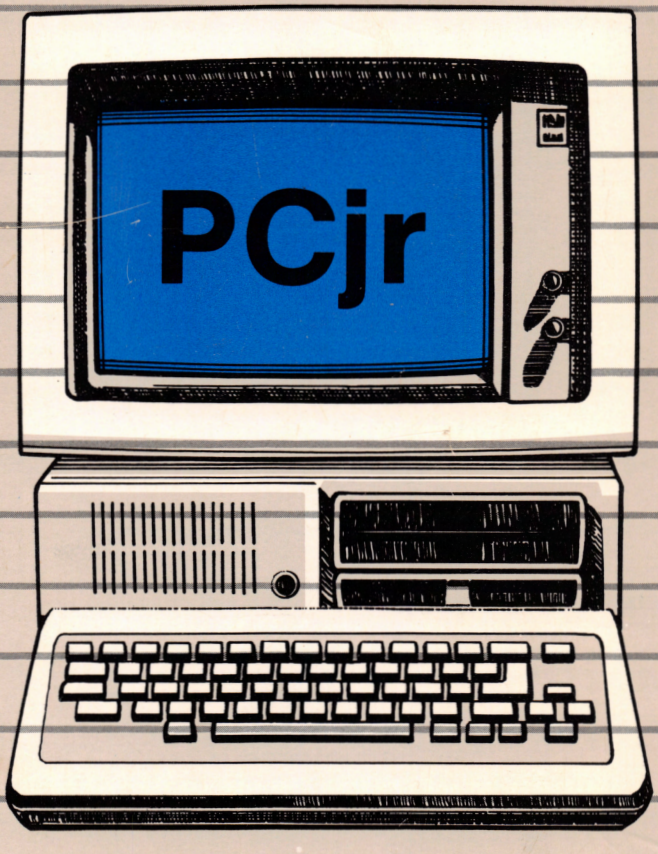
IBM PCjr®

FOR STUDENTS

USER'S HANDBOOK



WEBER SYSTEMS, INC. STAFF

IBM PCjr®
for Students

IBM PCjr®
for Students

by
WSI Staff

Weber Systems, Inc.
Cleveland, Ohio

The authors have exercised due care in the preparation of this book and the programs contained in it. The authors and the publisher make no warranties either express or implied with regard to the information and programs contained in this book. In no event shall the authors or publisher be liable for incidental or consequential damages arising out of the furnishing, performance, or any information and/or programs.

Microsoft BASIC®, Multiplan™, and Adventure™ are trademarks of the Microsoft Corporation; Homeward™ is a trademark of Sierra On-Line, Inc.; Intel 8088®, 4004®, 8008®, 8080®, 8085®, and 8088/8086® are registered trademarks of Intel Corporation; Home Budget™ is a trademark of Howe Software; Visicalc® is a registered trademark of Visicorp, Inc.; SN 76489A™ is a trademark of Texas Instruments; The following are trademarks of IBM Corporation. This book has been neither authorized or endorsed by IBM Corporation.

IBM PC*jr*®
IBM PC®
IBM PC XT®
IBM PC*jr* BASIC™
Cassette BASIC™
IBM Serial Adapter Cable™
IBM Parallel Printer Attachment™
IBM PC*jr* Attachable Joystick™
IBM PC*jr* Memory and Display Expansion Board™
Graphics Definition Language™
Cartridge BASIC™
IBM Graphics Printer™
IBM PC*jr* Internal Modem™
IBM DOS 2.1®
IBM Compact Printer™

Published by:
Weber Systems, Inc.
8437 Mayfield Road
Cleveland, Ohio 44026

For information or translations and book distributors outside of the United States, please contact WSI at the above address.

IBM PC*jr*® for Students

Copyright© 1984 by Weber Systems, Inc. All rights reserved under International and Pan-American Copyright Conventions. Printed in United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopy, recording, or otherwise without the prior written permission of the publisher.

Library of Congress Cataloging in Publication Data
Main entry under title:

IBM PC*jr* for Students

Includes index
1. IBM PC*jr* (Computer) — Programming — Juvenile literature. 2. Basic (Computer program language) — Juvenile literature. I. I. Weber Systems, Inc. II. Title: I.B.M. P.C.*jr* for Students.
QA76.8.I2593126 1984 001.64'2 84-50843
ISBN 0-938862-25-1

Typesetting and Layout: Tina Koran, Maria Stamoulis, Beth Cammarn, and Irma Schaeffer

Contents

Introduction 18
Purpose of this Book 18
Introduction 19
How to Use this Book 19

Section 1 Background 20

Lesson 1. What is a Computer? 22
Introduction 23
Computers Defined 23
Types of Computers 27
History of Computers 28
Questions 36

Lesson 2. Introducing the IBM PC*jr* 40
Introduction 41
PC*jr* — Entry Model and Enhanced Model 41
System Unit 43
Intel 8088 Microprocessor 44
RAM 48
ROM 48
Connectors for External Devices 49
Expansion Slots 49
Keyboard 52
Power Supply/Transformer 53
Video Display 54
Questions 56

Lesson 3. Peripherals and Add-on Devices 58
Introduction 59
Diskette Drive 59
Disk Operating System 62
Tracks and Sectors 62
Diskette Capacity 64
PC*jr* Disk Drive Operation 64
Printers 66
Sending Data from the PC*jr* to the Printer 68
IBM Printers 69
Joysticks 70
PC*jr* Keyboard Cord 71
PC*jr* Memory and Display Expansion 72
PC*jr* Internal Modem 73
Cassette Recorder/Players 74
PC*jr* Cartridges 75
Questions 77

Section 2 Discovery 80

Lesson 4. Introduction to BASIC 82
Introduction 83
Programming Language High Level, Machine, and Assembly 83
History of BASIC 84
Compiled vs. Interpreted Languages 84
Cassette and Cartridge BASIC 85
Types of Software 86
Questions 88

Lesson 5. Getting Started with BASIC 90
Introduction 91
PC*jr* System Start-Up Review 91
BASIC Start-Up 92
PC*jr* Start-Up Without the Diskette Drive 93
PC*jr* Start-Up With the Diskette Drive 95
Adjusting the Screen 101
Warm Boot 101
PC*jr* Keyboard 102
Fn Key 104
Alt Key 105
Ctrl Key 107
Editing Keys 107
Enter Key 108

Conclusion ____ 111
Questions ____ 112

Lesson 6. Your First Program ____ 114
Introduction ____ 115
Immediate and Program Modes ____ 115
Writing and Entering a Program ____ 117
Running a BASIC Program ____ 124
Clearing the Screen ____ 125
Listing the Program ____ 126
Erasing the Program ____ 126
Questions ____ 127

Lesson 7. How the PC*jr* Works ____ 130
Introduction ____ 131
BASIC Program Entry ____ 133
Running a BASIC Program ____ 135
Clearing the Screen with a BASIC Program in Memory ____ 141
Listing the BASIC Program ____ 142
Erasing the BASIC Program from Memory ____ 143
Questions ____ 144

Lesson 8. RUN, LIST, AUTO, RENUM, and DELETE ____ 148
Introduction ____ 149
RUN ____ 149
LIST ____ 151
AUTO — Automatic Generation of Line Numbers ____ 154
RENUM — Renumbering Program Lines ____ 158
DELETE — Deleting Program Lines ____ 164
Questions ____ 168

Lesson 9. Editing Your BASIC Program ____ 172
Introduction ____ 173
Line Entry Editing ____ 173
Editing Keys ____ 174
Cursor Up ____ 179
Cursor Down ____ 180
Cursor Right ____ 181
Cursor Left ____ 183
Delete ____ 184
Ins ____ 185
Backspace ____ 187
Esc Key ____ 188
Tab ____ 189
Fn-End ____ 191

Fn-Home 192
Fn-Break 193
Ctrl-Fn-End 194
Ctrl-Fn-Home 195
Ctrl-PgDn 196
Ctrl-PgUp 197
EDIT Command Entry 199
Cursor Movement Editing 201
Questions 204

Lesson 10. Saving and Loading BASIC Programs 208
Introduction 209
Cassette Recorder 209
Cassette Recorder Installation 209
Saving a BASIC Program on Cassette 212
Loading a BASIC Program from Cassette 214
Disk Drive 216
Formatting a Diskette 216
Displaying the Diskette Directory 219
Saving and Loading BASIC Programs on Diskette 220
Erasing a File from a Diskette 221
Questions 223

Lesson 11. Data Types and Variables in BASIC 226
Introduction 227
String Data 227
String Data Examples 227
ASCII 229
Numeric Data 229
Integers 230
Fixed-Point Numbers 230
Floating-Point Numbers 231
Hexadecimal Numbers 232
Octal Numbers 232
Numeric Precision 232
Variables 234
Variable Names 235
Assigning Values to Variables with the LET Statement 236
How Variables are Processed 237
Questions 244

Lesson 12. Operators 248
Introduction 249
Arithmetic Operators 249

Addition (+) 249
Exponentiation (^) 251
Floating Point Division (/) 252
Integer Division (\) 254
Modulo Arithmetic (MOD) 255
Multiplication (*) 257
Negation (-) 258
Subtraction (-) 259
Order of Evaluation 259
Mixing Variable Types in Arithmetic Expression 261
Relational Operators 261
Logical Operators 262
NOT 263
AND 263
OR 264
XOR 264
Order of Evaluation 265
Questions 267

Lesson 13. Outputting Data 270
Introduction 271
PRINT 271
PRINT USING 274
Formatting Characters 275
Numeric Formatting Characters - Pound Sign (#) 275
Numeric Formatting Characters - Decimal Point (.) 277
Numeric Formatting Characters - Plus Sign (+) 278
Numeric Formatting Characters - Minus Sign (-) 279
Numeric Formatting Characters - Comma (,) 281
Numeric Formatting Characters - Dollar Sign ($) 282
Numeric Formatting Characters - Asterisk (*) 284
Numeric Formatting Characters - Exponential Notation (^^^^) 286
String Formatting Characters - Ampersand (&) 287
String Formatting Characters - Backslash (\) 288
String Formatting Characters - Exclamation Point (!) 290
Literals 291
Formatting Functions: TAB,SPC,SPACE$ 292
TAB 292
SPC 293
SPACE$ 295
Questions 297

Lesson 14. Inputting Data 300
Introduction 301
INPUT 301
INPUT$ 302
LINE INPUT 304
Questions 306

Lesson 15. Conditional, Branching, and Looping Statements 308
Introduction 309
IF THEN 309
GOTO 312
GOSUB,RETURN 313
Conditional Statements with Branching 315
FOR,NEXT 316
WHILE,WEND 319
Questions 322

Lesson 16. Tables and Arrays 326
Introduction 327
Subscripted Variables and Arrays 327
Tables 331
DIM 333
OPTION BASE 335
DATA and READ 336
ERASE 338
Questions 339

Lesson 17. Numeric and Math Functions 342
Introduction 343
SIN,COS,TAN,ATN 343
SQR 345
INT 346
FIX 347
ABS 348
SGN 349
EXP 350
LOG 351
CINT,CSNG,CDBL 352
Questions 354

Lesson 18. String Functions 356
Introduction 357
String Concatenation 357
LEFT$ 358

RIGHT$ 359
MID$ 360
STR$ and VAL 361
CHR$ and ASC 363
INSTR 364
LEN 365
STRING$ 366
Questions 368

Lesson 19. Other Functions and User-Defined Functions 372
Introduction 373
FRE 373
POS 374
PEEK 375
POKE 376
RND 376
RANDOMIZE 380
SCREEN 381
User-Defined Functions 382
Questions 384

Lesson 20. Introduction to Graphics 386
Introduction 387
Pixels 387
Text Mode 388
Low Resolution Graphics 390
Medium Resolution Graphics 391
Screen 1 392
Screen 4 393
Screen 5 394
High Resolution Graphics 394
SCREEN 395
WINDOW 398
VIEW 398
Questions 400

Lesson 21. Graphics Statements 402
Introduction 403
Absolute and Relative Form 403
PSET and PRESET 404
LINE 406
CIRCLE 408
DRAW 410

Vertical and Horizontal Movements 411
Diagonal Movements 412
Scaling Factor 413
M 414
B 415
N 416
C 416
A 417
TA 418
P 419
X 421
GDL Commands and Variables 422
PAINT 423
Questions 430

Lesson 22. Introduction to Sound 432
Introduction 433
Sound Generators 433
SOUND 434
BEEP 434
SOUND 435
PLAY 437
Notes 438
L 438
MB and MF 440
Articulation 440
N 441
O 442
P 443
T 444
V 445
Dotted Notes 445
Changing Octaves 446
X 447
NOISE 448
Questions 451

Lesson 23. Programming Techniques 454
Introduction 455
Top-Down Design 455
Delay Routine 464
Menu-Driven Programming 465

Techniques using Variables 468
Initializing Variables 468
Flags 471
Significant Variable Names 472
Questions 473

Section 3 Applications 476

Lesson 24. Applications for Mathematics 478
Introduction 479
Algebra 479
Geometry 481
Trigonometry 486

Lesson 25. Applications for Science 494
Introduction 495
Chemistry 495
Conversion Program 495
Percentage Composition 499
Limiting Reagent 500
Ideal Gas Law 503
Physics 506
Motion 506
Work 511

Lesson 26. Writing Papers and Reports 512
Introduction 513
Overview of HomeWord 514
Starting HomeWord 517
Creating a Title Page 518
Enter Title Page Information 518
Change Top Margin 519
Define Line Spacing 521
Center Lines 522
Main Text of the Paper 523
Start a New Page 524
Enter Main Text 525
Redefine Margins 526
Justify Text 527
Indenting the Quote 528
Single Space Quote 530
Footnote the Quote 531

Footnotes 533
Placing Footnotes at the End of the Paper 533
Placing Footnotes at the Bottom of the Page 537
Bibliography Page 540
Page Numbers 543
Appendix A. ASCII Codes 545
Appendix B. BASIC Reserved Words 548
Appendix C. Answer Key 549
Index 553

Introduction

Purpose of this Book

At this point, you may be wondering, "Why should I study computers?". An understanding of computers is important for every member of society for a number of reasons. Two of the most important reasons are:

- ☐ Computers have influenced almost every aspect of our daily lives.
- ☐ Computers will have an even greater influence in the future.

Our primary intention in this book will be to help you become **computer literate.** Literate is derived from the Latin term, litteratus, which can be translated as learned.

People who are computer literate have an understanding of computers that allows them to use computers as tools in their everyday lives. All tools are extensions of parts of our bodies. For example, an automobile is an extension of our feet, a telescope is an extension of our eyes, and a scuba tank is an extension of our lungs. The automobile allows us to travel farther and faster than our feet could carry us. A telescope allows us to see into outer space — far beyond

the reaches of our eyesight. A scuba tank allows us to swim underwater for long periods of time. This would be impossible using our body's natural breathing organs.

A computer is an extension of the human mind. Computers can be used to accomplish repetitive tasks that would otherwise occupy our thought processes. In most instances, these tasks can be accomplished more quickly and more accurately by using computers. Computers then are tools that we can use to extend the power of our minds as well as to free our minds from being occupied with repetitive and generally boring tasks.

Acknowledgements

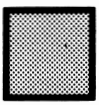
We gratefully acknowledge Jeanette Mahrer of IBM, for her invaluable assistance. We also wish to thank Bob Baker, Computerland, for his assistance and cooperation.

How to Use this Book

This book consists of a series of 26 lessons, which are designed to be studied in sequence. These lessons include explanations, examples, exercises, and step-by-step instructions to be used while working with the PC*jr*. Answers to many of the exercises can be found in Appendix C.

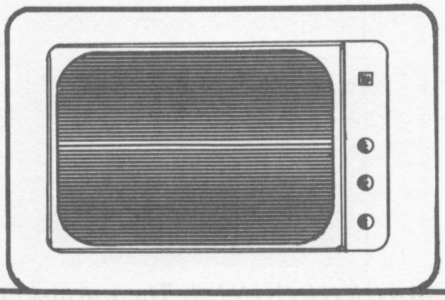
Terms which you may find unfamiliar are presented in bold. These terms will be defined in subsequent paragraphs.

Section 1



Section 1 is simply titled "BACKGROUND". The purpose of the three lessons which make up Section 1 is to help you gain a solid understanding of the general concepts of computers and computing. This knowledge will provide a solid foundation upon which you can begin building your understanding of computers. We will attempt to accomplish the following goals in Section 1:

- ☐ *Gain a general overview of how a computer functions*
- ☐ *Apply this knowledge to the PCjr so as to gain an understanding of how it functions*
- ☐ *Gain an understanding of the terms used in computing such as bit, byte, input, output, microprocessor, RAM, ROM, etc.*
- ☐ *Gain an understanding of the various PCjr components, peripheral devices, and add-on devices*
- ☐ *Gain an understanding of the history and evolution of computing devices*



What is a Computer?

lesson 1

Lesson Goals

- *Define the term computer*
- *Gain a general understanding of how a computer functions*
- *Become familiar with the history and evolution of computing devices*

Introduction

In this lesson, a general overview of computers will be presented. A basic definition of a computer will be provided along with a condensed history of computing. This lesson is the most general in this guide and is meant to remove the confusion that many people feel when discussing computing. Hopefully, this lesson will stimulate your interest in computers and will help you feel comfortable while learning about computers.

After reading this lesson, you should understand how computers function on a general level. Computers are powerful problem solving tools. They are very important in today's society and can only become more so in the future. An understanding of computers is one of the most valuable assets a person can have in today's world.

Computers Defined

The word "computer" often frightens people. Don't let it frighten you. A computer can be defined as an electronic machine that uses a **program**, or set of logical instructions, to process data at a relatively high rate of speed. A computer generally performs the following five functions:

- ☐ Input
- ☐ Storage
- ☐ Control
- ☐ Processing
- ☐ Output

These functions are depicted in figure 1.1.

Data must first be **input**, or sent, to the computer. Input can be defined as the process of sending data into the computer. Input might be in any one of a number of different forms including:

- ☐ Keyboard entries
- ☐ Readings from a measuring device such as a thermometer

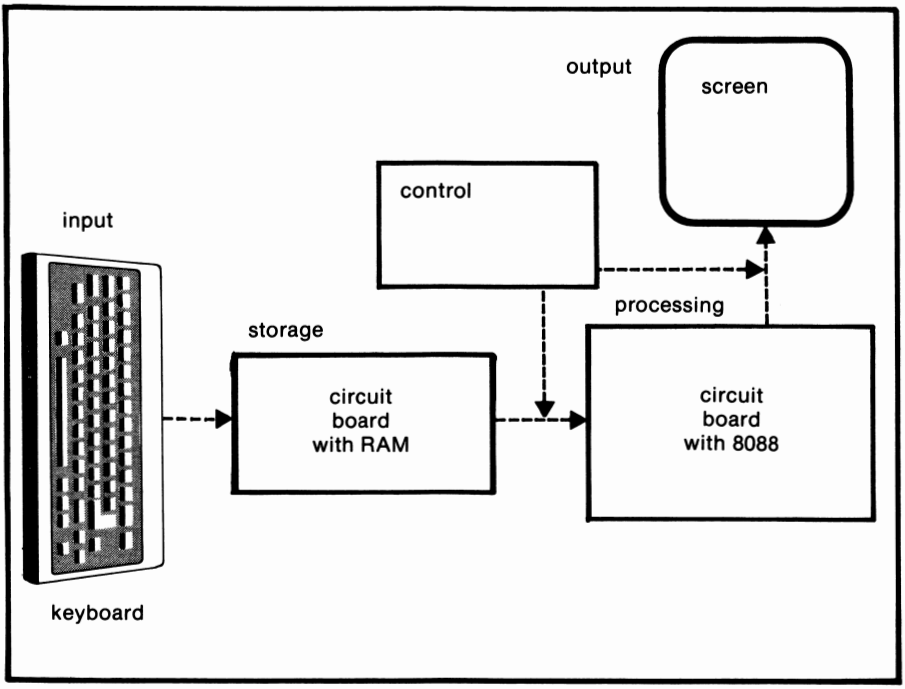


Figure 1.1. Computer information processing functions

- ☐ Data read from a punched card
- ☐ Data received from a communications device such as a modem
- ☐ Data entered by pressing a joystick or game controller

Once data has been entered, it is stored, generally in magnetic memory, for future use. Data is input and stored in binary code. Binary codes consist of 1's and 0's and use the base two numbering system. For example, the following number in binary format:

$$100_2$$

would be represented as 4 in the decimal or base 10 numbering system.

Once data has been input and stored, it can be **processed** by the computer. Data processing can be defined as the various operations performed on data according to the instructions issued by the program. Examples of data processing operations include:

- ☐ Calculating a formula
- ☐ Sorting a list of names into alphabetical order
- ☐ Calculating the class average on a mid-term exam
- ☐ Comparing two values

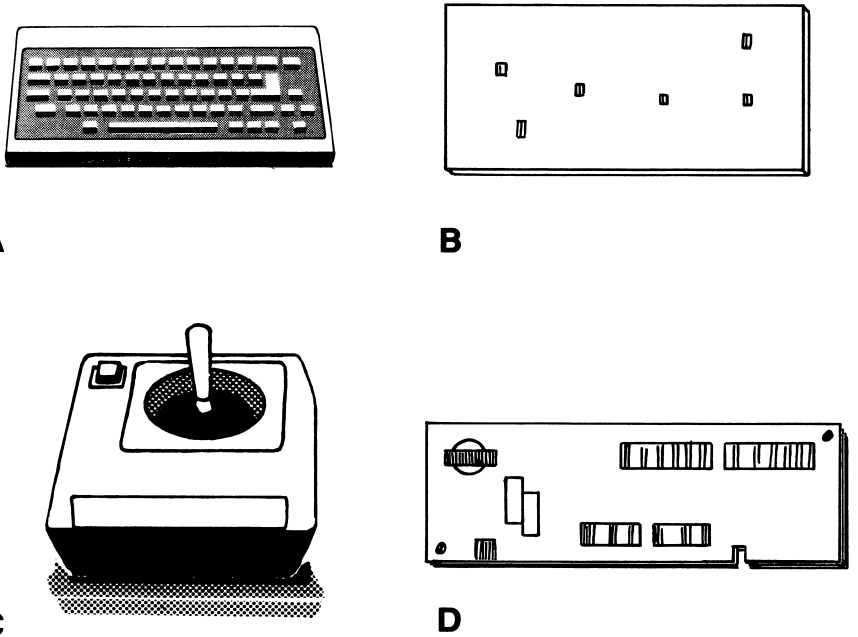


Figure 1.2. Input examples

a, Keyboard entry; *b*, Punched card; *c*, Game controller; *d*, Modem

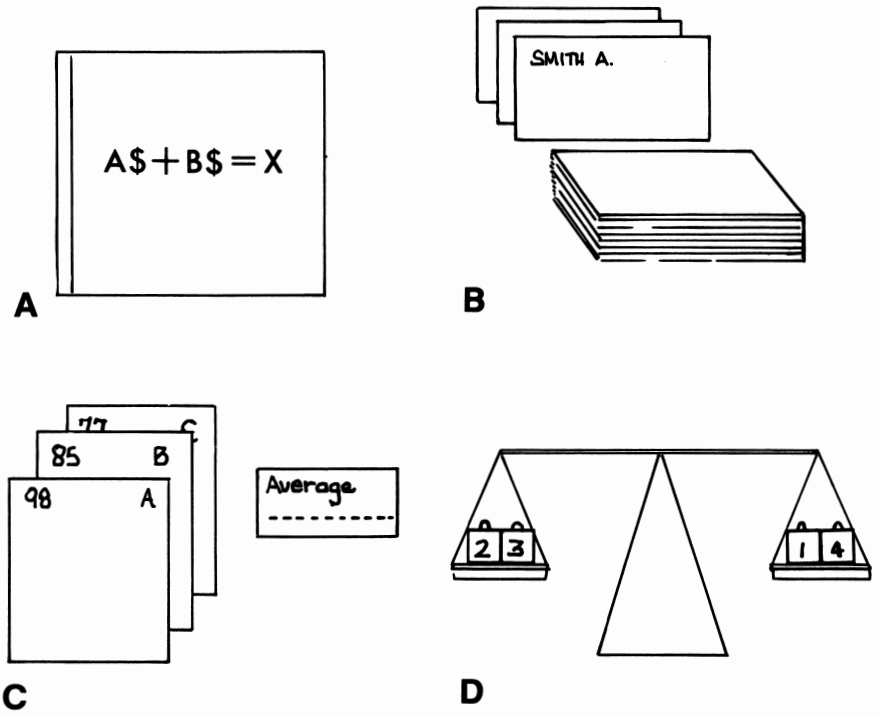


Figure 1.3. Data processing examples

a, Calculating a formula; *b*, Sorting a list of names; *c*, Calculating an average; *d*, Comparing two values

Notice that a number of the computer's processing activities are the same as a calculator's. However, the computer has one important additional feature — **the ability to make decisions** based upon instructions stored in memory. These instructions constitute the computer's **control** function. For each activity that it undertakes, the computer must be given an instruction.

Once data has been processed, it can be **output**. Output can be defined as the process of transmitting data which has been processed by the computer. Data can be output in a number of different forms including:

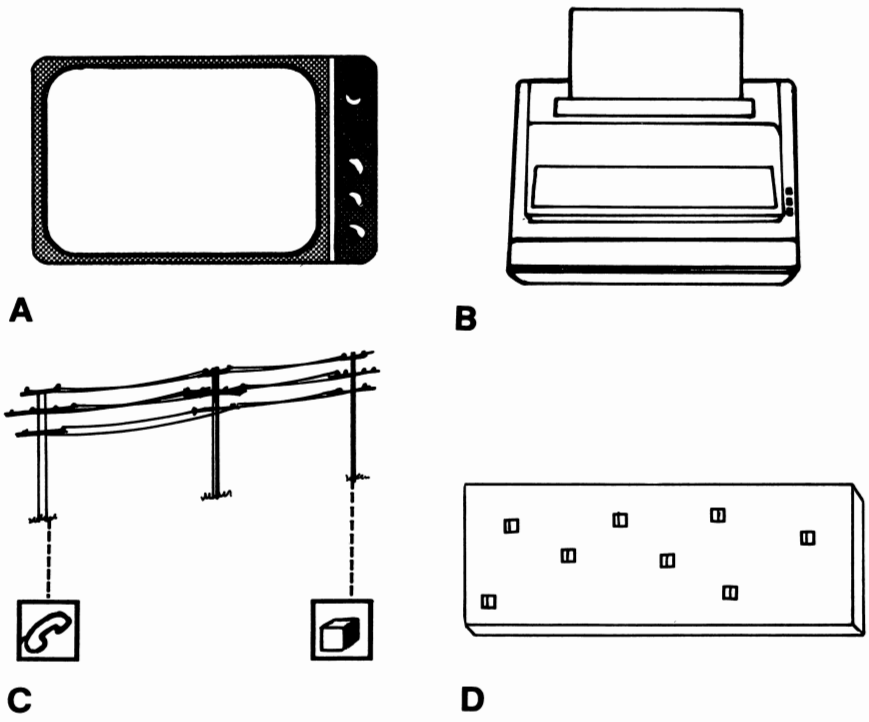


Figure 1.4. Output examples

a, Data output to the video display; *b*, Data output to the printer; *c*. Data transmitted via phone lines; *d*, Data output to a punched card

- ☐ A display on a computer video-display
- ☐ A printed report
- ☐ Codes which are communicated over telephone lines to other computers.

Types of Computers

Computers can be grouped according to three general classifications:

- ☐ Analog computers
- ☐ Digital computers
- ☐ Hybrid computers

An analog computer can be defined as a device that uses a physical quantity (generally electric current or voltage) to solve mathematical problems. In an analog system, the physical phenonemon being represented is simulated by the computer. A household thermostat is a good example of an analog computer. The temperature is represented by an electric current. If this current falls below a predetermined level, a signal will be sent to the heating plant.

Digital computers represent data in binary form. Groups of binary signals are used to represent numbers, characters, and symbols.The IBM PC*jr* is a digital computer. In this book, we will be discussing digital computers.

Hybrid computers are designed with both digital and analog characteristics. One example of the hybrid computer is a numerically controlled machine.

History of Computers

Computing devices can be traced back in history some 500 years to the abacus. The abacus consists of rows of parallel rods or wires upon which are mounted sliding blocks or beads. An abacus is depicted in figure 1.5. The beads are divided into two sections by a bar. The top section of the abacus has either one or two beads. These represent either 0 and 5 depending on their position on the rod.

Each rod on the bottom section contains four or five beads, each of which represents a single unit. The individual bars each represent a significant digit of a number. The least significant digit is indicated by the rod at the extreme right of the abacus.

Pascal's digital adding machine was a significant development in the history of computing devices. Blaise Pascal was a French mathematician, scientist, and philosopher. In 1642, when he was about twenty years old, Pascal designed and built a mechanical calculating machine to help him keep track of the accounts in his father's business. This was the forerunner of the modern desktop calculator. It consisted of a mechanical gear system which could add and subtract numbers containing as many as eight digits.

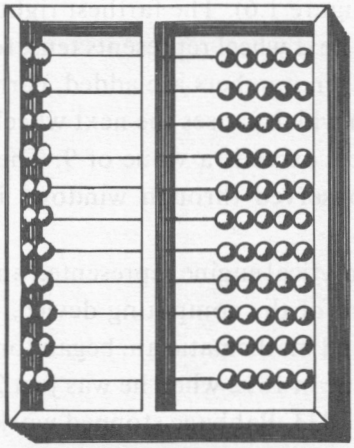
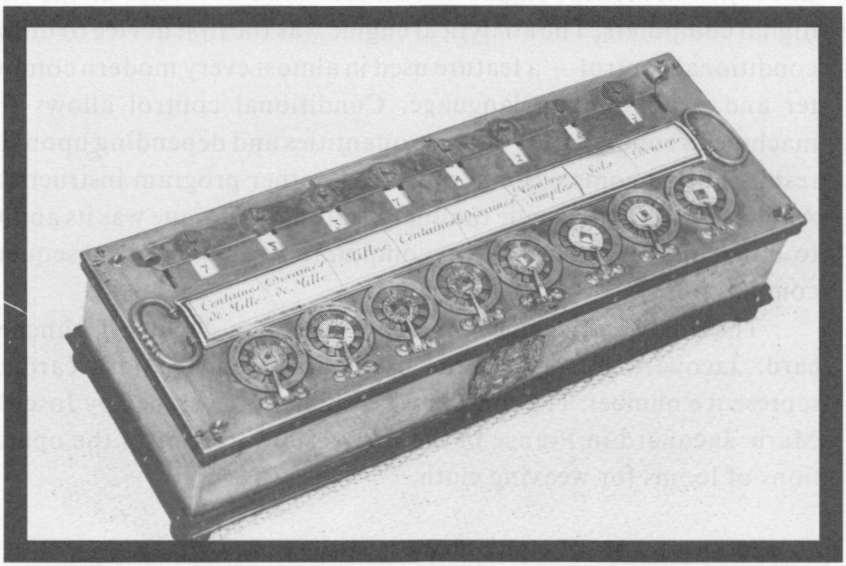


Figure 1.5. Abacus



"Courtesy of International Business Machines Corporation"

Figure 1.6. Pascal's arithmetic machine

Numbers are entered using eight dial wheels on the top side of the machine (see figure 1.6). The farthest right wheel represents the units position, the next wheel represents tens, followed by hundreds, thousands, etc. When numbers are added, carrying is accomplished by the gear system which causes the next wheel to turn by one when the preceding wheel exceeds a value of 9. The value at each wheel position can be observed through windows in the machine's top cover.

Babbage's analytical engine represented another significant step in the development of the computing device. Charles Babbage, an English inventor and mathematician, began formulating the idea for his analytical engine in 1812 when he was just 21 years old. Twenty-one years later, in 1833, Babbage stopped work on the project when the British government cut off funds for it. Babbage's analytical engine was displayed at the International Exposition of 1862 and can be viewed today at the Science Museum in South Kensington, London.

The analytical engine was a revolutionary machine in its time. Babbage's analytical engine was in fact the forerunner of the first digital computers. The analytical engine was the first device to utilize conditional control — a feature used in almost every modern computer and programming language. Conditional control allows the machine to compare two or more quantities and depending upon the results of that comparison, branch to another program instruction. Another feature that made the analytical engine unique was its ability to utilize the results of its own computations as data in subsequent computations.

The analytical engine was controlled by a Jacquard punched card. Jacquard punched cards use a hole punched in a card to represent a number. The concept was originally developed by Joseph-Marie Jacquard in France in the early 1800's to control the operations of looms for weaving cloth.

The analytical engine used three sets of punched cards for outputting data and instructions. These were:

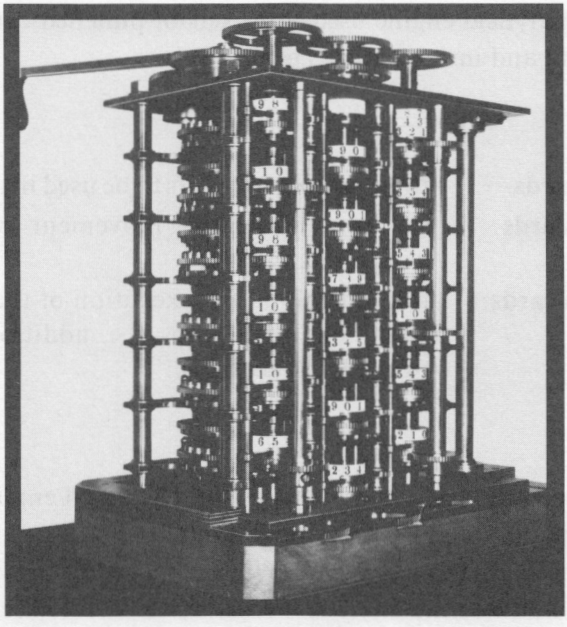
Number cards	for inputting numbers to be used in the problem
Directive cards	for controlling the movement of numbers within the machine
Operation cards	for controlling the execution of the machine's arithmetic operations (i.e. addition, subtraction, multiplication)

The two primary components of the analytical engine were the storage unit and the mill. The storage unit consisted of groups of 50 counter wheels that could store 1000 numbers each consisting of 50 digits. The mill was the analytical engine's calculator section.

Hollerith's tabulating machine represented another milestone in the development of the digital computer. Herman Hollerith was born in 1860 and educated at Columbia University in New York City. It was at Columbia that Hollerith began studying tabulating systems. After Hollerith graduated from Columbia at age 20, he took a job with the U.S. Census Office.

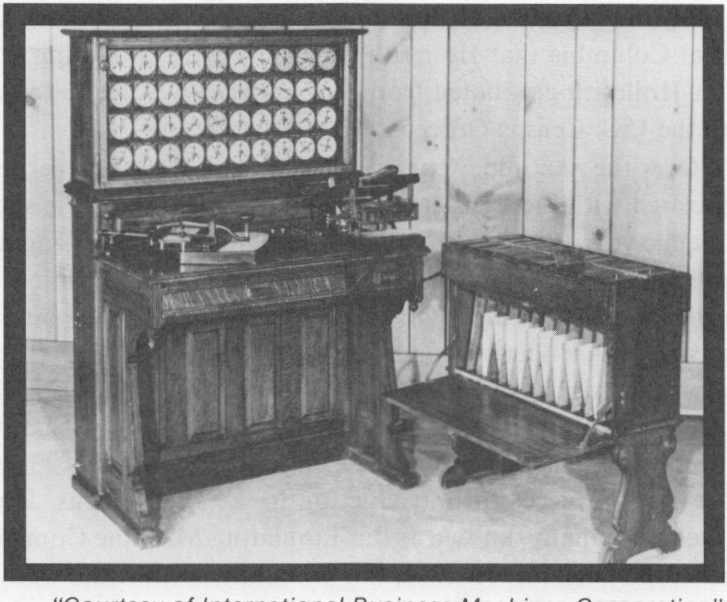
Over the next nine years, Hollerith led a varied career. He was associated with the Census Office until 1883, although he spent the academic year 1882-1883 as an instructor in mechanical engineering at the Massachusetts Institute of Technology. In 1883, Hollerith took a position with the Patent Office for approximately one year. For the next six years, Hollerith worked on the development of his tabulating equipment, which was designed to calculate census data. Hollerith received patents on his machine in 1889. The tabulating machine was utilized with great success by the Census Office in the 1890 census.

After its successful introduction in the 1890 census, Hollerith formed a company, known as the Tabulating Machine Company, to market the machine. In 1911, this company became the Computer-Tabulating-Recording Company, which later became the International Business Machines Corporation, or IBM.



"Courtesy of International Business Machines Corporation"

Figure 1.7. Babbage's analytical engine



"Courtesy of International Business Machines Corporation"

Figure 1.8. Hollerith's tabulating machine

The Hollerith tabulating machine made use of 6⅝ by 3¼ inch punched cards to input data. Holes punched in the cards were used to represent various characteristics such as age, sex, etc. of the respondents to the census. Hollerith designed one machine for punching the cards and another for sorting them. Each card would be run underneath a set of contact brushes. These would complete an electric circuit if a hole was present in the card. When the circuit was completed, a counter would be advanced.

The great advantage to Hollerith's tabulating machine was that the cards could be assembled into a large deck and sorted according to any single characteristic. This allowed census data to be analyzed quickly and accurately.

The Harvard Mark I* represented the next significant advancement in the development of the computer. The Mark I was developed by Howard Aiken and a group of IBM engineers under the direction of Clair D. Lake. While studying physics as a graduate student at Harvard University, Aiken studied punched card calculating machines similar to those manufactured by IBM and analyzed the alterations that would be required to adapt these for scientific calculations. Aiken analyzed that the following adaptations would have to be incorporated in punched card calculating machines to make them practical for scientific applications:

- ☐ Ability to handle negative as well as positive numbers.
- ☐ Abiltiy to handle the various complex functions involved in scientific calculations.
- ☐ Ability to perform calculations without the need for human interaction.
- ☐ Ability to perform calculations in their natural mathematical order.

* Also known as the IBM Automatic Sequence Controlled Calculation

Aiken's ideas came to the attention of the Watson Computing Bureau at Columbia University in New York City, which was funded by the IBM Corporation. In 1939, Aiken and a group of IBM engineers began designing a new computing device. The Harvard Mark I was completed in 1944. It measured 50 feet in length and was 8 feet high. The computer was controlled by a paper tape which contained the machine's instructions. These instructions consisted of three parts. One instruction contained information regarding where the data to be operated on was stored. Another determined what operation was to be performed on that data. The third determined where the result of the operation was to be stored.

The Mark I could calculate using the basic arithmetic operators (addition, subtraction, multiplication, division) and could also calculate logarithms, exponentials, and sines. The Mark I could handle negative as well as positive numbers. Data was input into the Mark I using punched cards. Data was output either on punched cards or on an electric typewriter.

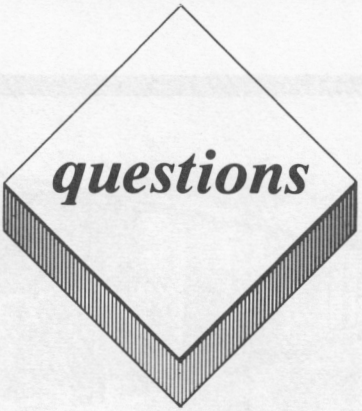
The ENIAC (an acronym for Electronic Numerical Integrator and Calculator) was the first general purpose, electronic digital computer. Earlier computing devices, including the Mark I, had been electromechanical rather than electronic. Recall from the definition of computers on page 23 that we defined the computer as an electronic device. With its electronic design, the ENIAC could perform calculations at speeds 1000 times faster then electromechanical computers.

The ENIAC was designed by J. Presper Eckert and John W. Mauchly of the University of Pennsylvania. In 1946, Eckert and Mauchly formed their own company for the purpose of designing and building computers. They sold the firm to Remington Rand Corporation in 1959. Eckert and Mauchly also designed and built the Univac I for use by the United States Bureau of Census. The Univac I represented yet another significant advancement in the evolution of the electronic digital computer. The Univac I was the first computer that could handle alphabetic information with the same competence as numeric information. The Univac I was probably the first electronic digital computer to be used widely in the business and government environments. It was the forerunner of our modern computers.



"Courtesy of International Business Machines Corporation"

Figure 1.9. Harvard Mark I



True or False

1. The IBM PC*jr* is an analog computer.
2. A digital computer uses binary data.
3. Information displayed on a computer's video display screen could be described as input.
4. Punched cards are used both for input and output in computer systems.
5. Babbage's analytical engine was the first electronic digital computer.
6. Punched cards were first used with Babbage's analytical engine.
7. Hollerith's Computer-Tabulating-Recording Company was the forerunner of the IBM Corporation.

Multiple Choice

1. Input can be defined as:
   - A. The process of storing data for future use.
   - B. Performing calculations on stored values.
   - C. The process of sending data to the computer.
   - D. Transmitting data which has been processed by the computer.
   - E. None of the above

2. The following are examples of data processing:
   A. Calculating 5+7–3÷8
   B. Sorting a list of names in alphabetical order
   C. Deciding which value is greater, 27.82 or 26.98
   D. All of the above
   E. None of the above

3. The first computing device to incorporate conditional control was:
   A. Pascal's adding machine
   B. ENIAC
   C. Harvard Mark I
   D. Babbage's analytical engine
   E. None of the above

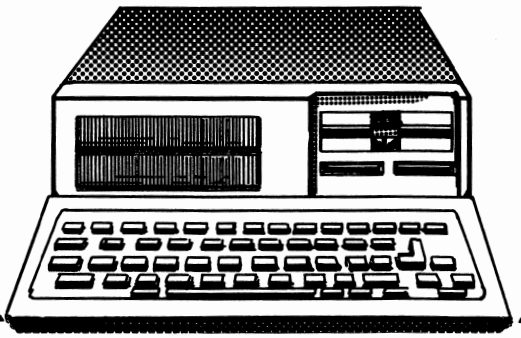
4. Punched cards were first used with:
   A. Pascal's adding machine
   B. Jacquard's looms
   C. Babbage's analytical engine
   D. Hollerith's tabulating machine
   E. None of the above

5. The Harvard Mark I was developed by:
   A. Howard Aiken and IBM engineers
   B. Presper Eckert and John W. Mauchly
   C. Harry Hollerith
   D. All of the above
   E. None of the above

6. The first general purpose electronic, digital computer was:
   A. Univac I
   B. Harvard Mark I
   C. Hollerith's tabulating machine
   D. ENIAC
   E. None of the above

Essay

1. What is your definition of the term — computer?
2. What five functions are performed by computers?
3. Describe each of these five functions.
4. What is your definition of conditional control?



Introducing the IBM PCjr

lesson 2

Lesson Goals

- *Gain an understanding of each of the PCjr's four primary components*
- *Learn the meaning of computing terms related to these peripheral and add-on devices*

Introduction

At this point, we have provided a broad definition of computers, outlined the development of modern digital computers, and discussed the importance of becoming computer literate in today's society. With this background, we are ready to begin our journey towards computer literacy.

For this journey, we will use the IBM PC*jr* computer. In this lesson, we will introduce the PC*jr* and its various components.

PCjr -- Entry Model and Enhanced Model

Two different models of the PC*jr* are available, the entry model and the enhanced model. These are pictured in figures 2.1 and 2.2. The major difference between the two models is that the enhanced model includes a floppy diskette drive, while the entry model does not. Also, the enhanced model allows 80 characters to be displayed on each line on the screen, while only 40 characters can be displayed per line on the entry model.

The IBM PC*jr* includes four basic components. These are as follows:

- ☐ System unit
- ☐ Keyboard
- ☐ Power supply/transformer

We will discuss each of these in the following sections.

System Unit

The system unit is pictured in figures 2.3 and 2.4. It contains the electronic circuitry that performs the PC*jr*'s control and processing functions, as well as some of its storage function. Most of this circuitry is located on the PC*jr*'s system board. The system board is depicted in figure 2.5. The major components located on the system board are as follows:

- ☐ Intel 8088™ microprocessor
- ☐ 64K RAM
- ☐ 64K ROM
- ☐ Connectors for external devices
- ☐ Expansion slots

Each of these will be discussed in the following sections.

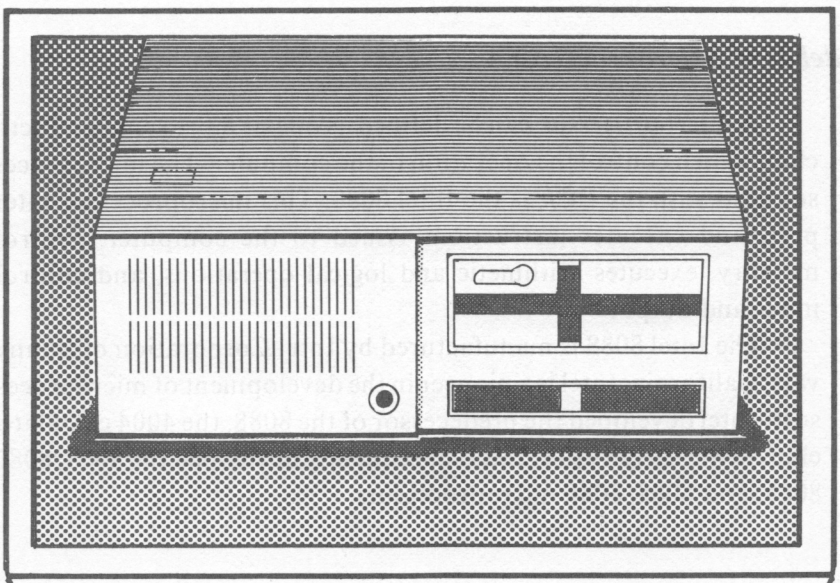


Figure 2.3. System unit (front view)

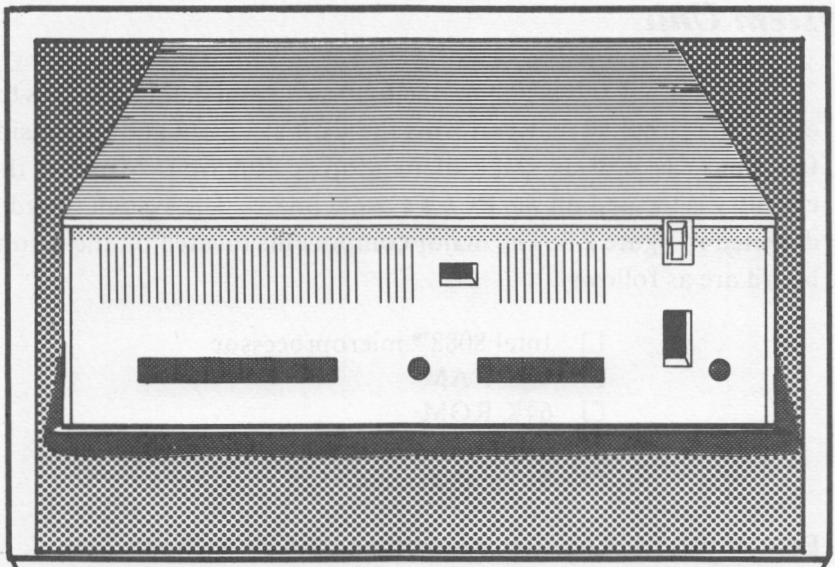


Figure 2.4. System unit (rear view)

Intel 8088 Microprocessor

A **microprocessor** can be defined as one or more semiconductor chips which control the operation of the computer. The microprocessor used with the PC*jr* is the Intel 8088. This microprocessor interprets and executes instructions issued to the computer, controls memory, executes arithmetic and logical operations, and controls input and output.

The Intel 8088 is manufactured by Intel Corporation of Sunnyvale, California. Intel is a pioneer in the development of microprocessors. Intel developed the predecessor of the 8088, the 4004 calculator chip, in the early 1970's. Intel subsequently developed the 8008, 8080, 8085, and 8088/8086 microprocessors.

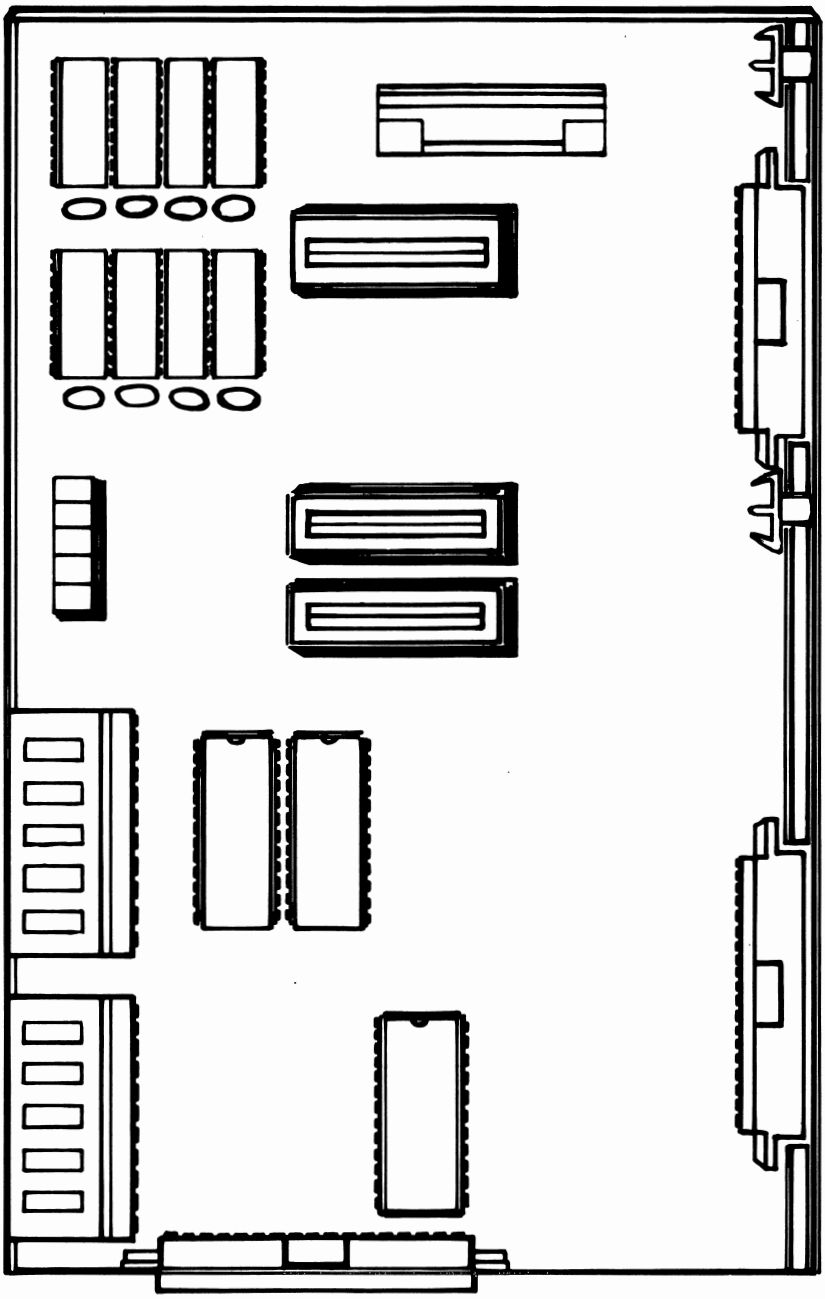


Figure 2.5. PC*jr* system board

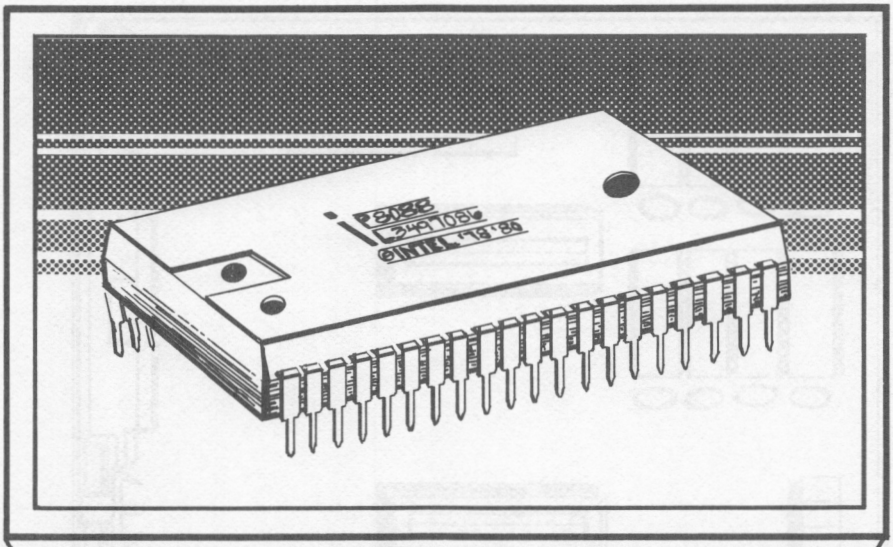


Figure 2.6. Intel 8088 microprocessor

Microprocessor logic is based upon the **bit.** A bit is the basis of all information storage within the computer. A bit consists of a simple switch that can consist of either of the two binary states, on or off. If the switch is on, a value of 1 is returned. If the switch is off, a value of zero is returned.

Bits are often separated into groups of eight. These groups of 8 bits are known as a **byte.** A byte is required to represent a single character (i.e. letter, number, or symbol). Generally, bytes are processed by the computer in groups of 2.

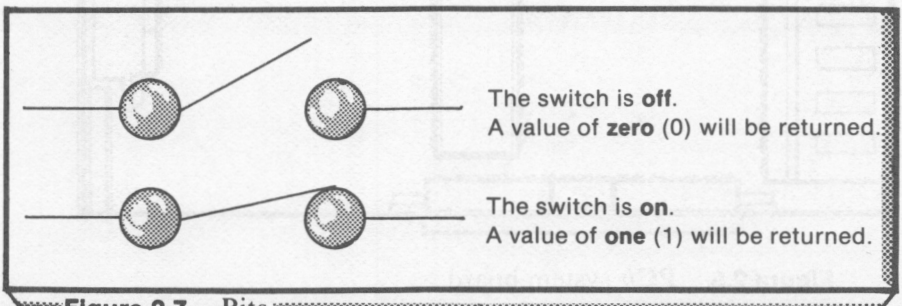


Figure 2.7. Bits

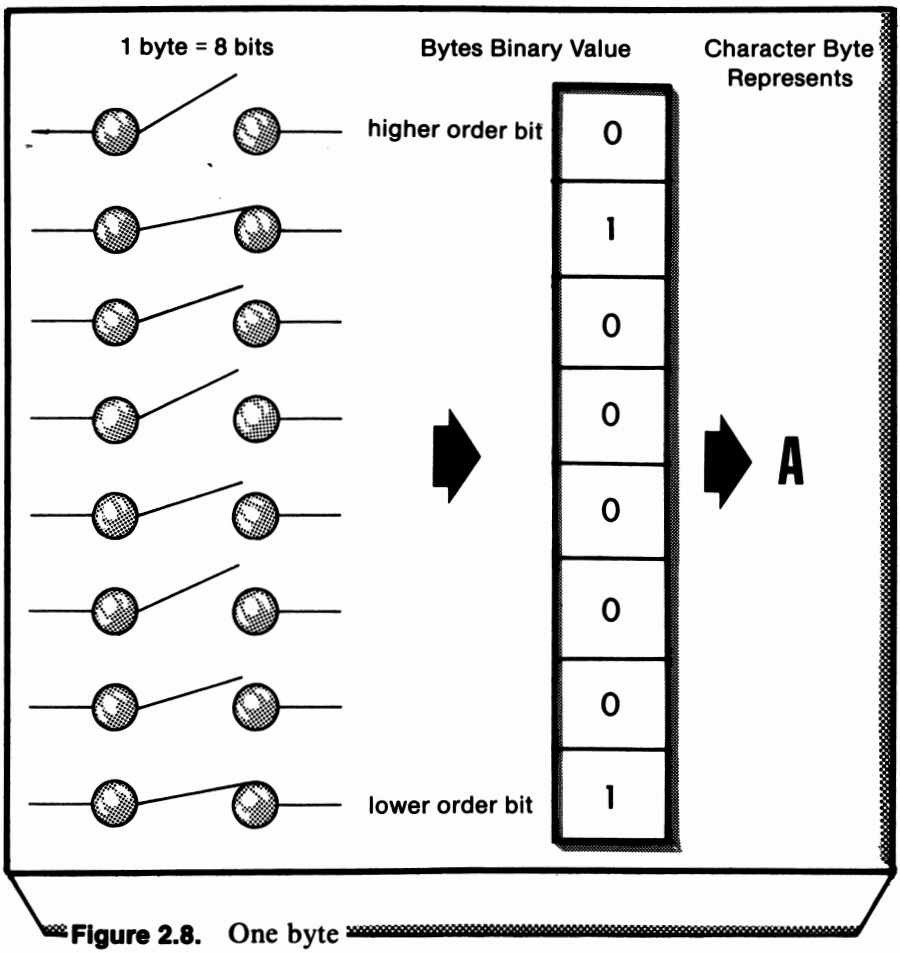


Figure 2.8. One byte

Most 8-bit microprocessors can only **address** (or work with) 65,536 bytes at any one time. Even though this number appears large, a 30 page document would fill this area. Most 16-bit microprocessors can address from 65,536 to 16 million bytes of memory. Moreover, 16-bit microprocessors process data at a speed from 2 to 10 times faster than 8-bit microprocessors.

The Intel 8088 is a 16-bit microprocessor. One of the main advantages the IBM PC*jr* has over other home computers is the power provided by its use of the 8088 CPU.

RAM

RAM is a type of **memory** contained on the PC*jr*'s system board. The terms memory and storage can be used synonymously. Data can be sent to and held in memory in its binary form.

RAM is an abbreviation for random access memory. Access is random because any individual storage location in RAM can be accessed immediately. This allows data to be read from or written to RAM very quickly. Any data stored in RAM will be lost when the PC*jr*'s power is shut off. A RAM chip is shown in figure 2.9.

ROM

ROM is an abbreviation for read-only memory. Information is stored in ROM in a permanent or semi-permanent manner. This data can be read from ROM, however it cannot be changed during computer operation. If the power to the PC*jr* is shut off, the data stored in ROM will remain there. Individual memory locations in ROM are accessed in a random manner.

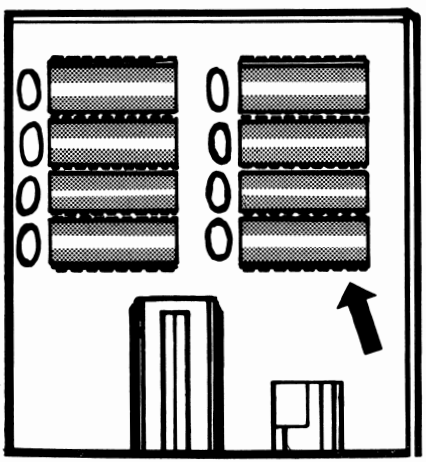


Figure 2.9. RAM chip

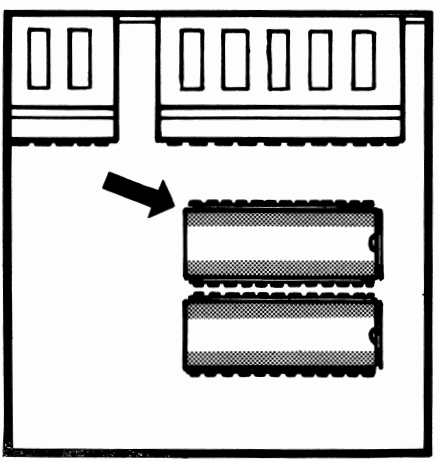


Figure 2.10. ROM chip

Connectors for External Devices

Notice from our expanded view of the rear of the system board in figure 2.9, that the connection ports in the rear of the system unit attach to the system board. The various connector ports are depicted in figure 2.11.

Expansion Slots

Notice the 4 expansion slots depicted in figure 2.5. One expansion slot (see *a* in figure 2.5) is utilized by the PC*jr*'s power supply board. This board is connected to the power supply/transformer. The PC*jr*'s on/off switch is installed on this board. The power supply board is depicted in figure 2.12.

The expansion slot denoted by *b* in figure 2.5 is used for installation of the PC*jr* Memory and Display Expansion Board. This board is shown in figure 2.13.

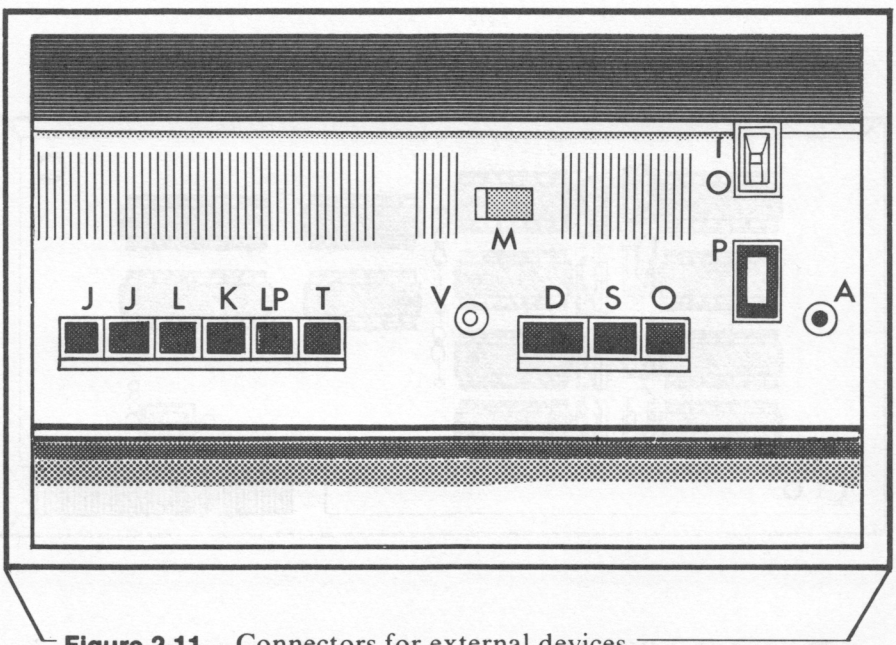


Figure 2.11. Connectors for external devices

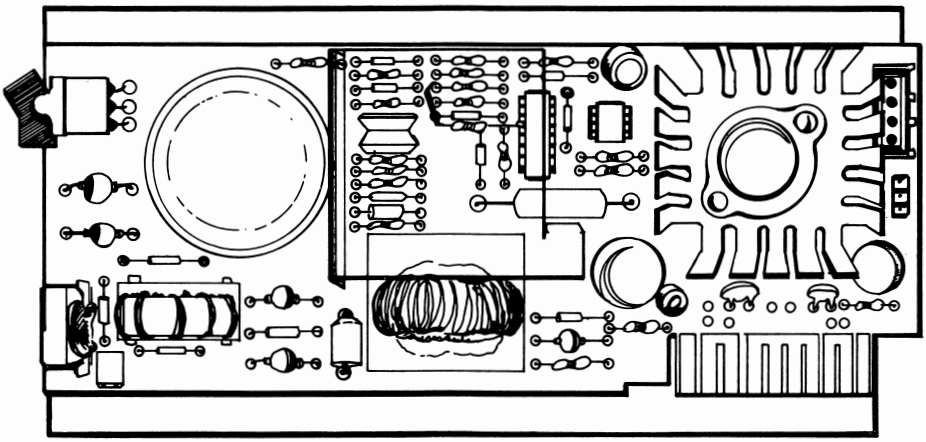


Figure 2.12. PC*jr* power supply board

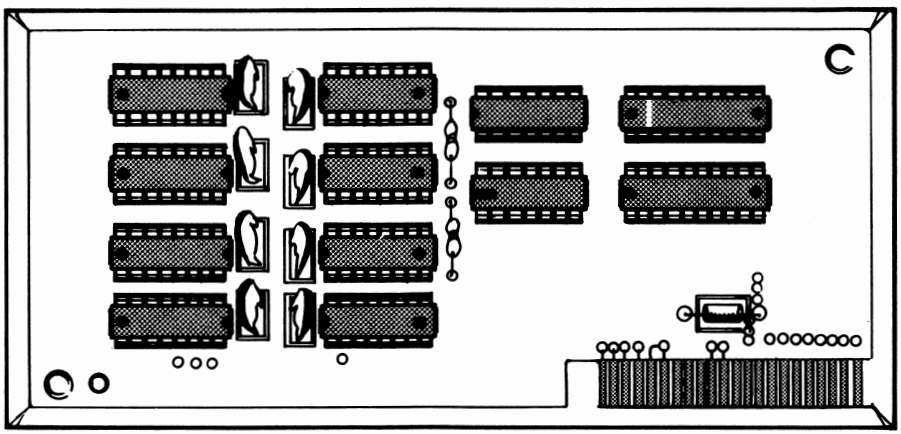


Figure 2.13. PC*jr* Memory and Display Expansion Board

The expansion slot denoted by *c* in figure 2.5 is used to install the optional Internal Modem. This board is pictured in figure 2.14.

The expansion slot denoted by *d* in figure 2.5 is used to install the disk controller board. This board is connected to the PC*jr*'s optional diskette drive. It is depicted in figure 2.15.

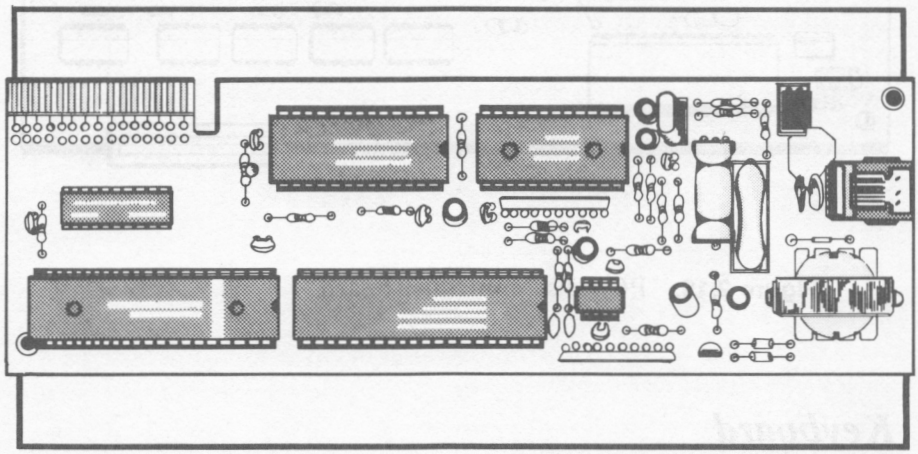


Figure 2.14. PC*jr* Internal Modem

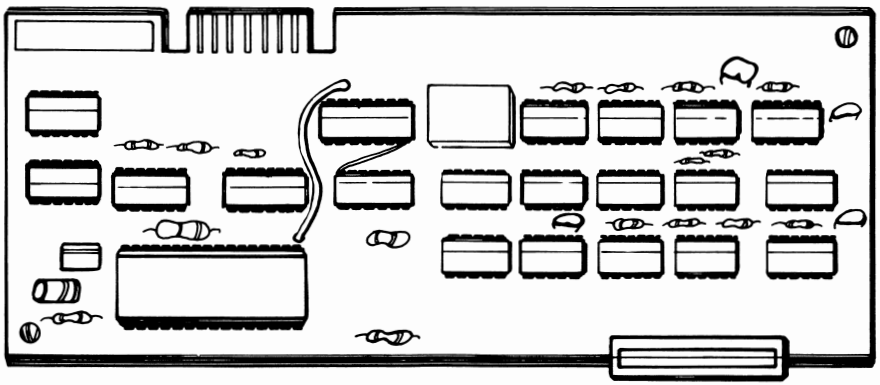


Figure 2.15. PC*jr* disk controller board

Keyboard

The PC*jr*'s keyboard allows the user to communicate with the system unit. Unlike most personal computers, the PC*jr* keyboard does not necessarily have to be connected to the system unit with a cable. The PC*jr* keyboard is cordless. Communication with the system unit is accomplished using an infrared optical link.

If more than one PC*jr* is being used concurrently in a room, it is necessary to connect the keyboard to the system unit using the optional keyboard connection cable. Otherwise, the keyboard can be used without the cable to communicate with the system unit. The advantage of cordless communications is especially useful in a classroom situation -- as the system unit can be placed in a central location and the keyboard can be passed around the classroom. The effective range of cordless keyboard communication is approximately 20 feet.

The PC*jr* keyboard is powered by four AA batteries. It contains 62 **programmable*** keys arranged in a typewriter-like sequence. The individual keys are described in Lessons 5 and 9.

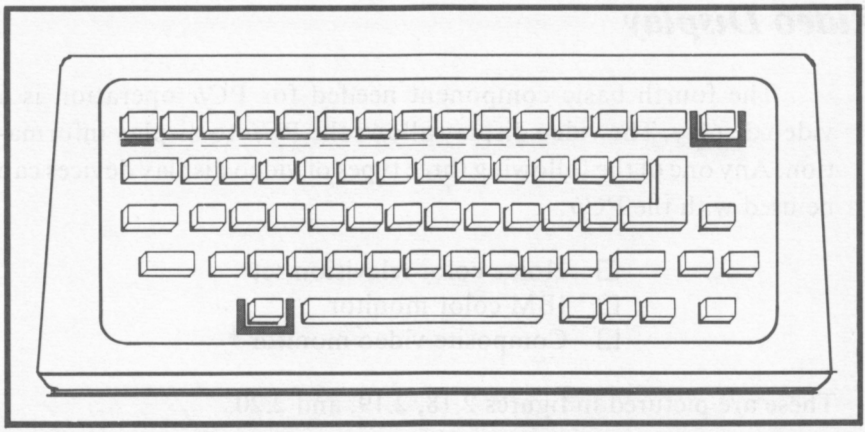


Figure 2.16. PC*jr* keyboard

Power Supply/Transformer

The PC*jr*'s power supply/transformer is not built into the system unit. Instead, it is housed in a separate unit which is attached to the rear of the system unit by a power cord.

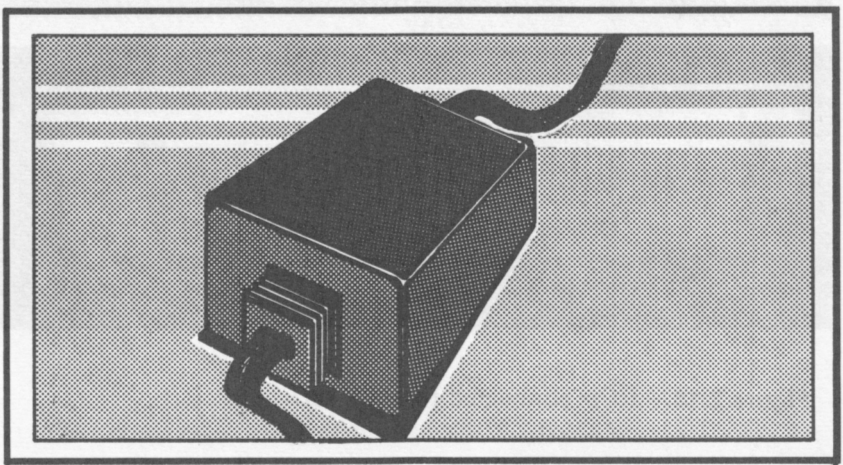


Figure 2.17. PC*jr* power supply/transformer

Video Display

The fourth basic component needed for PC*jr* operation is a video display. The video display allows the PC*jr* to display information. Any one of the following three types of video display devices can be used with the PC*jr*:

- ☐ Home color television set
- ☐ IBM color monitor
- ☐ Composite video monitor *

These are pictured in figures 2.18, 2.19, and 2.20.

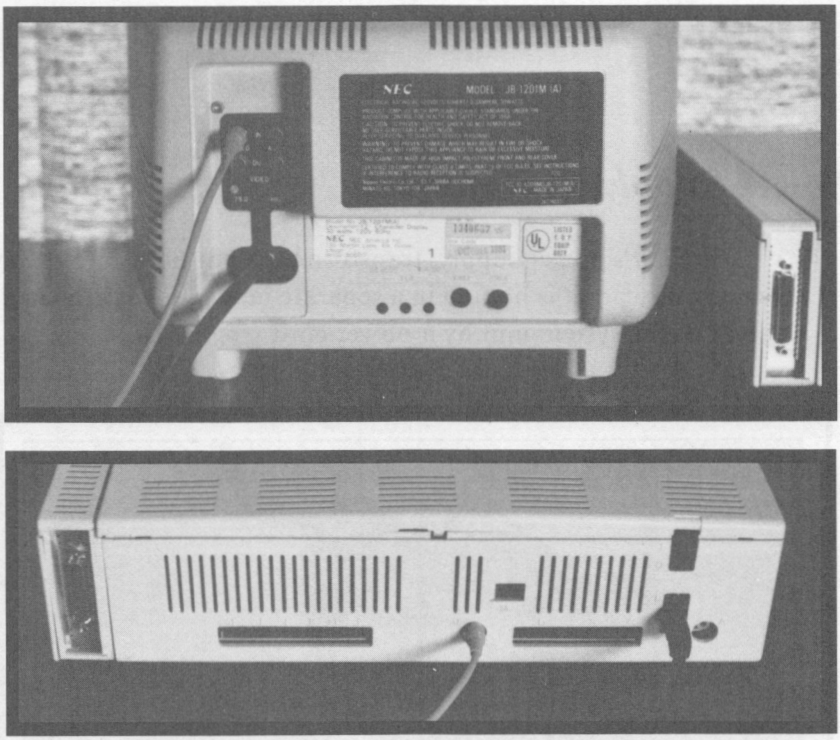


Figure 2.18. PC*jr* connected to a TV set

* A composite video monitor is a display device designed to display information transmitted by a computer.

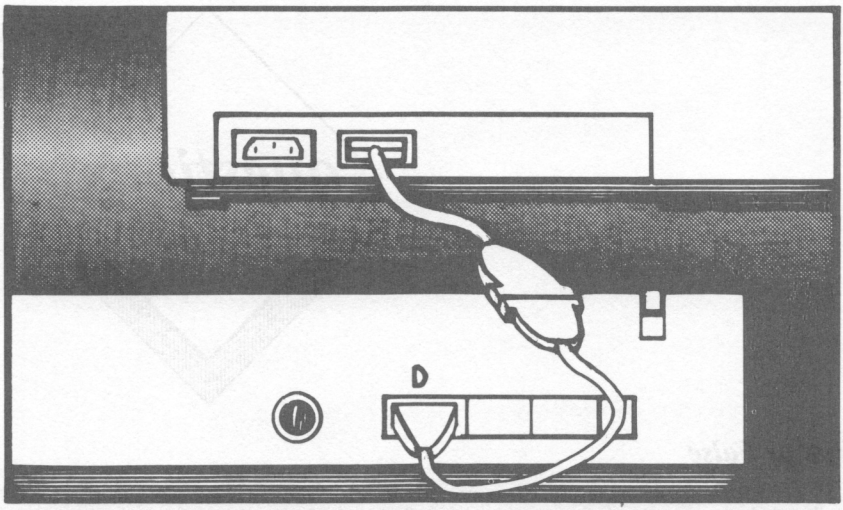


Figure 2.19. PC*jr* connected to an IBM color monitor

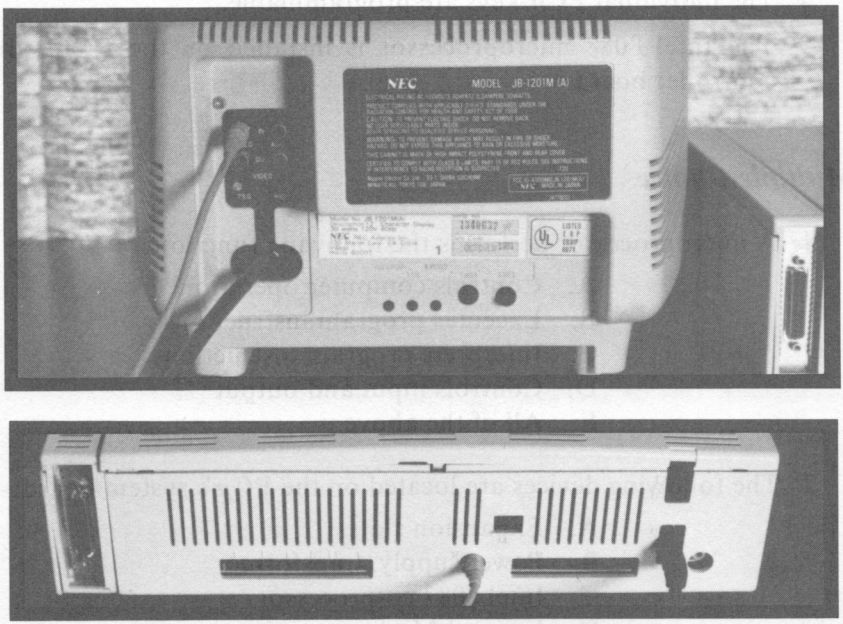
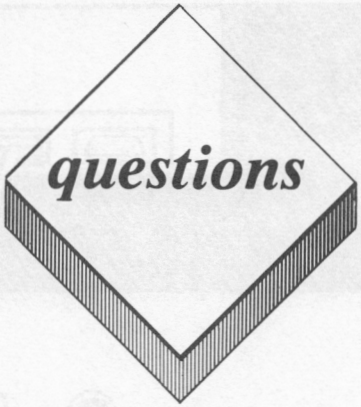


Figure 2.20. PC*jr* connected to a composite video monitor



True or False

1. A byte can only assume one of two values, 1 or 0.
2. A single bit can be used to represent an individual character.
3. A cable is not required for communications between the PC*jr*'s system unit and keyboard.
4. The individual PC*jr* keys are programmable.
5. The Intel 8088 microprocessor is installed on the PC*jr*'s disk controller board.

Multiple Choice

1. A microprocessor performs the following functions:
   - A. Controls computer operation
   - B. Executes program instructions
   - C. Interprets program instructions
   - D. Controls input and output
   - E. All of the above

2. The following devices are located on the PC*jr*'s system board:
   - A. Expansion slots
   - B. Power supply/transformer
   - C. Intel 8087 co-processor
   - D. Internal Modem
   - E. All of the above

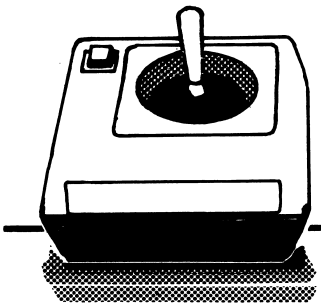
3. The expansion slots on the PC*jr* system board are used to install:
   A. Joysticks
   B. Printers
   C. Internal Modem
   D. ROM Cartridges
   E. None of the above

4. This type of memory loses the data stored in it when the PC*jr*'s power is turned off.
   A. ROM
   B. Diskette
   C. Cassette
   D. RAM
   E. None of the above

5. How much RAM is included on the PC*jr*'s system board?
   A. 64MB
   B. 64K
   C. 128K
   D. 32K
   E. None of the above

Essay

1. Define the terms RAM and ROM, noting especially the differences between the two.
2. Describe the various functions of a microprocessor.



Peripherals & Add-On Devices

lesson 3

Lesson Goals

- *Gain an understanding of the various peripheral and add-on devices available for the PCjr*
- *Learn the meaning of computing terms related to these peripherals and add-on devices*

Introduction

In the last lesson, we described the main components of the PC*jr*: the system unit, the keyboard, and the power supply/transformer. A number of **peripherals*** and add-on devices can be added to these basic components. These include:

- ☐ Diskette drive
- ☐ Printers
- ☐ Joysticks
- ☐ Keyboard cord
- ☐ Memory and Display Expansion device
- ☐ Internal Modem
- ☐ Cassette player/recorder
- ☐ Cartridges

Several of these devices were discussed briefly in lesson 2. We will discuss these in more detail in this lesson.

Diskette Drive

The diskette or disk drive is one of the most important parts of a computer system. Disk drives allow the storage of relatively large amounts of data and also offer relatively fast access to that data. Unlike RAM storage, when information is stored on a disk, the information is not lost when the computer is turned off. In other words, disks offer a permanent means of storing data. The addition of a diskette drive will greatly enhance the usefulness of the PC*jr*.

Data is stored on a diskette in magnetic form. The disk drive contains a device known as a **read/write head** which is used to read information from and write information to the diskette. The read/write head is depicted in figure 3.1.

---

* A peripheral can be defined as an auxiliary device which can be connected to a computer to perform some additional function.

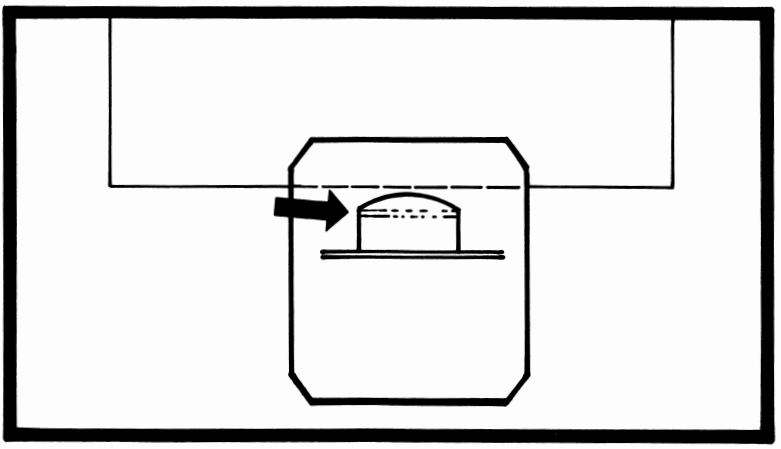


Figure 3.1. PC*jr* read/write head

With a disk drive, data can be read from or written to any particular position on the diskette. This is in contrast to magnetic tape data storage, where, in order to access a particular piece of data, all preceding data items must first be examined. The concepts of random and sequential data access are depicted in figure 3.2.

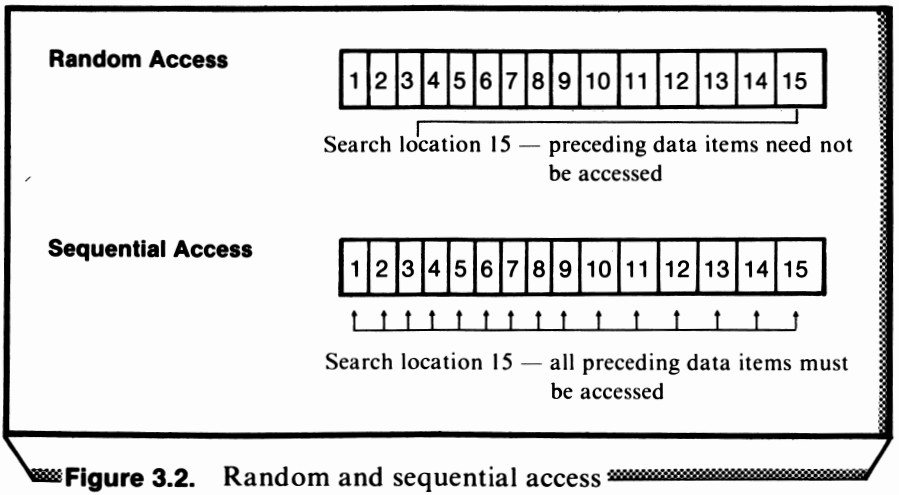


Figure 3.2. Random and sequential access

A diskette consists of a round vinyl disk which is stored inside of a plastic cover. The diskette is stored in a protective paper envelope which protects it from damage while it is being stored or handled. A 5¼ inch diskette and its protective paper envelope is pictured in figure 3.3.

When the diskette is inserted in a drive, the round vinyl disk inside the plastic cover is rotated. Notice the round hole in the middle of the diskette. This allows the disk drive to hold the diskette and to spin the vinyl disk inside the cover. Also, notice the oval shaped opening on the surface of the diskette's plastic cover. This opening provides an area where the disk drive's read/write head can read data from or write data to the disk surface.

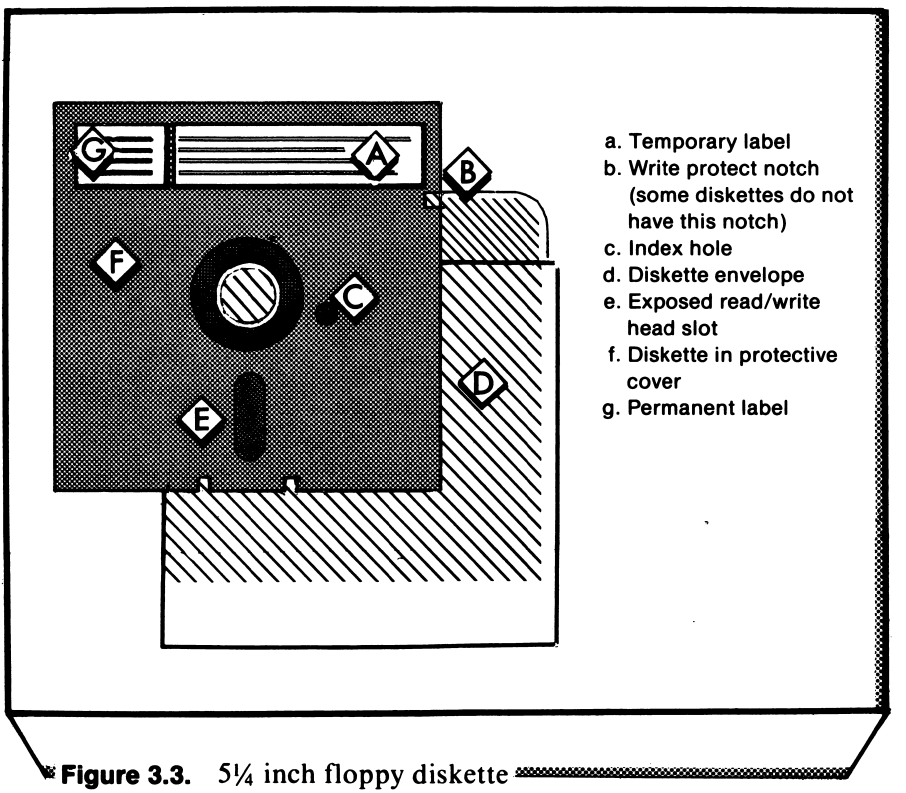


Figure 3.3. 5¼ inch floppy diskette

Disk Operating System

The disk operating system (DOS) is a group of programs which allow the user to manipulate information between the diskette drives, memory, and the video screen. The disk operating system used on the PC*jr* is DOS 2.1. This was developed by Microsoft Corporation and is a revised version of the earlier versions of DOS used on the IBM PC and PC XT.

Tracks and Sectors

The disk operating system partitions the diskette surface into imaginary areas known as tracks and sectors. By dividing the disk surface in this manner, DOS can identify specific locations on the diskette surface. This allows data to be located more easily.

Tracks may be visualized as a series of concentric bands on the diskette surface. This is illustrated in figure 3.4. DOS 2.1 divides the diskette surface into 40 individual tracks. Each of these tracks can be accessed by DOS.

To further reduce the amount of time required to search for a particular data item, DOS divides each track into sectors. Each track is divided into 9 sectors. Each individual sector holds 512 bytes of data. When DOS is given a track and sector number, it will only have to search 512 bytes to find a particular data item.

The process for locating a specific track on the diskette surface is fairly simple. The drive moves the read/write head to the specified track. Locating a particular sector is more difficult. The PC*jr* uses the soft sector method to locate a particular sector on the diskette surface.

An index hole is used in the soft sector method to locate individual sectors. The index hole is located just to the right of the large hole in the center of the diskette. The index hole is depicted in figure 3.3. As shown in figure 3.3, the index hole is only located on the diskette's plastic cover. However, another index hole is located on the actual diskette surface inside the plastic cover.

As the drive spins the diskette, the index hole on the diskette's surface passes underneath the hole in the protective envelope.

A light source inside the disk drive shines light onto the area of the diskette containing the index hole. When the index hole on the disk surface is aligned with the index hole in the protective envelope, the light will shine through to a sensor. The sensor will relay information on the location of the index holes, which can be used to calculate the various sector locations.

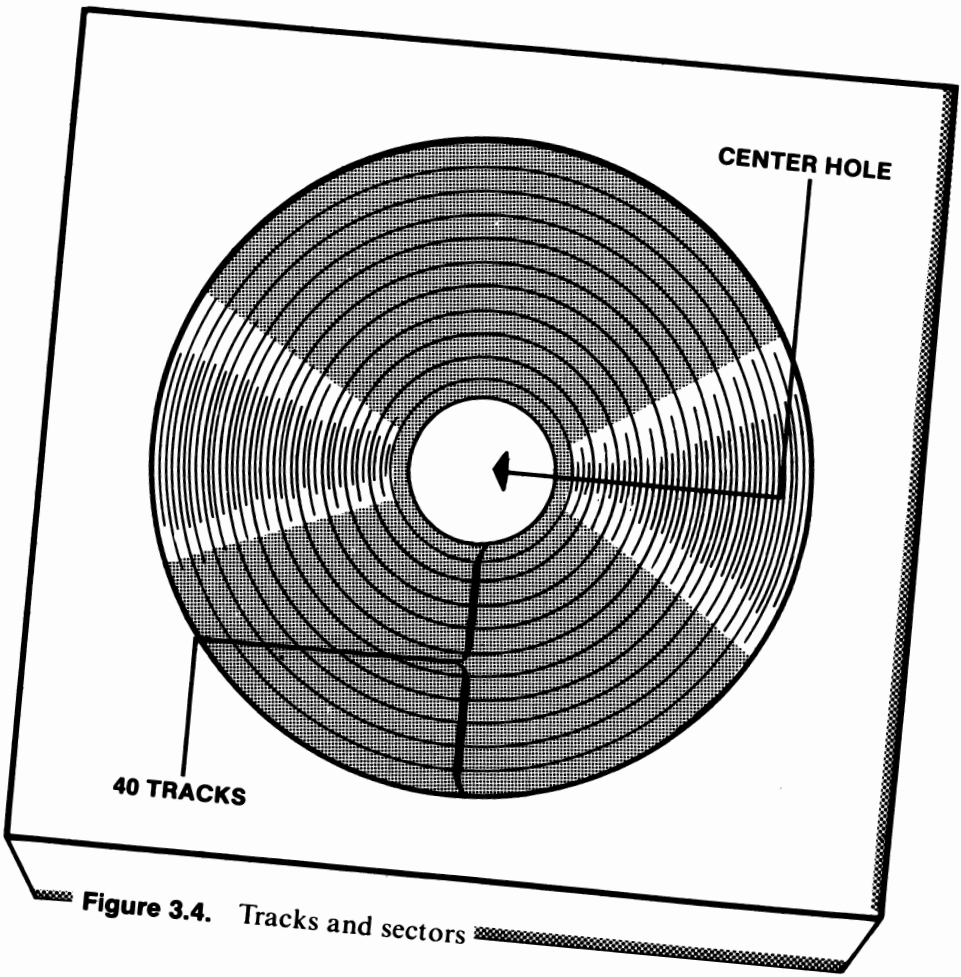


Figure 3.4. Tracks and sectors

Diskette Capacity

The amount of data which can be stored on a diskette is affected by two storage characteristics -- density and the number of sides to which data can be written. Density refers to the recording format used on the diskette. Typical recording formats are single density and double density. Single density diskettes have the capacity to store approximately 90K of data per side, while double density diskettes have a capacity of about 180K per side.

Floppy diskettes can be designed so that data can be written to only one side or to both sides. Diskettes which are designed to be written to one side are known as single-sided diskettes. Those which are designed to be written to both sides are referred to as double-sided diskettes. The PC*jr* uses double sided, double density diskettes (DS,DD) with a capacity of 360K.

PCjr Disk Drive Operation

Operation of the PC*jr* disk drive is simple. When a diskette is not inserted into the disk drive, the disk slot handle should be in the horizontal or open position. This is depicted in figure 3.5.

When a diskette is being inserted into the PC*jr*'s disk drive, the diskette label should be facing up. The side of the diskette containing the oval shaped opening should be inserted into the drive as shown in figure 3.6. Once the diskette has been inserted into the drive, the diskette slot handle should be rotated to the vertical or closed position. A diskette can be removed from the drive by merely reversing this procedure.

The PC*jr* disk drive has a small red light on its front cover. Whenever data is being read from or written to the disk surface, this lamp will light. The diskette should not be removed from the drive while this lamp is on.

If the disk drive isn't operating properly, check to see if the monitor is placed on top of the computer. The disk drive may not operate properly under these conditions.

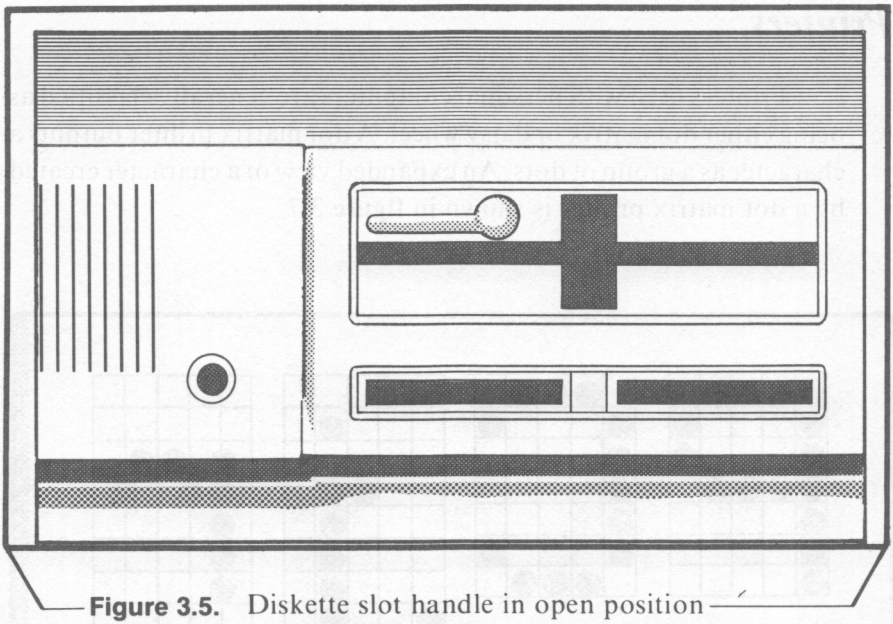


Figure 3.5. Diskette slot handle in open position

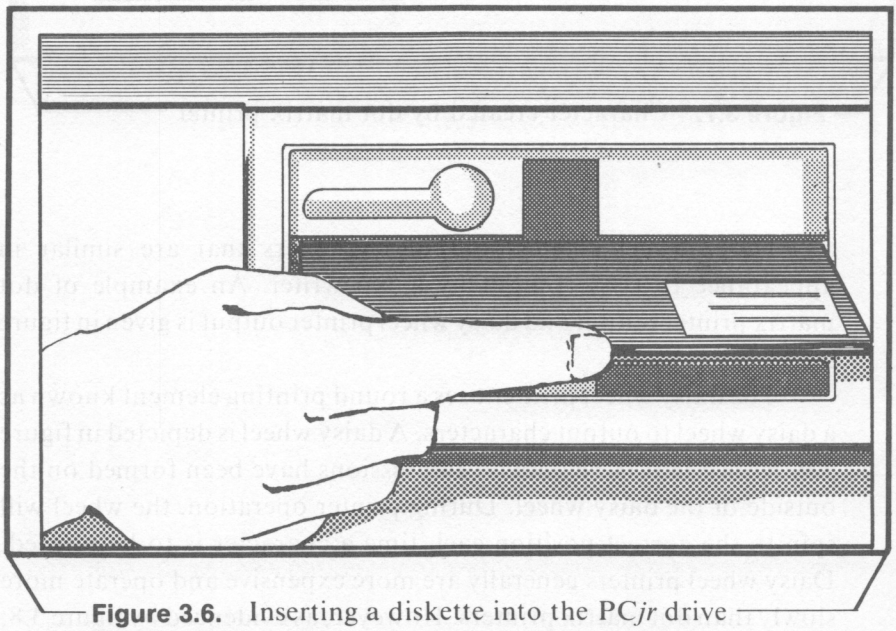


Figure 3.6. Inserting a diskette into the PC*jr* drive

Printers

Printers used with personal computers are generally classified as being either dot matrix or daisy wheel. A dot matrix printer outputs a character as a group of dots. An expanded view of a character created by a dot matrix printer is shown in figure 3.7.

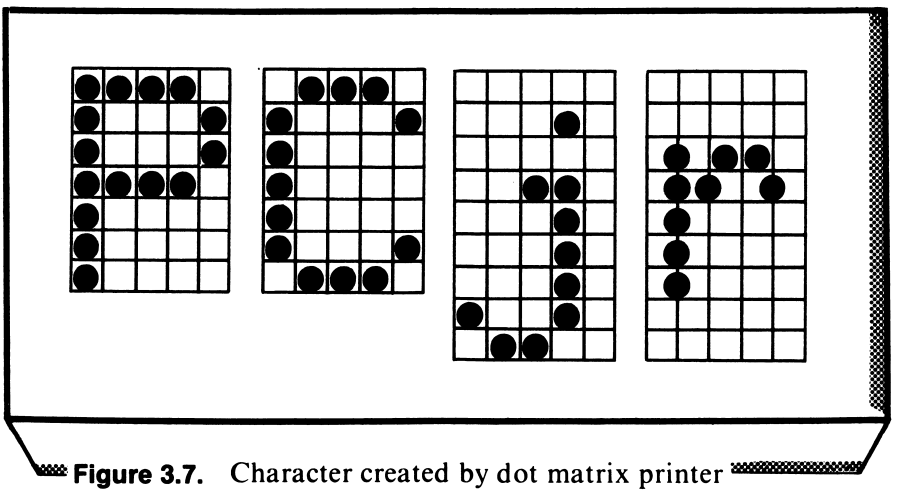


Figure 3.7. Character created by dot matrix printer

Daisy wheel printers output characters that are similar in appearance to those output by a typewriter. An example of dot matrix printer output and daisy wheel printer output is given in figure 3-8.

The daisy wheel printer uses a round printing element known as a daisy wheel to output characters. A daisy wheel is depicted in figure 3.9. Notice that the character impressions have been formed on the outside of the daisy wheel. During printer operation, the wheel will spin to the correct position each time a character is to be printed. Daisy wheel printers generally are more expensive and operate more slowly than dot matrix printers. However, as evidenced by figure 3.8, the quality of characters output by daisy wheel printers is higher than those output by dot matrix printers.

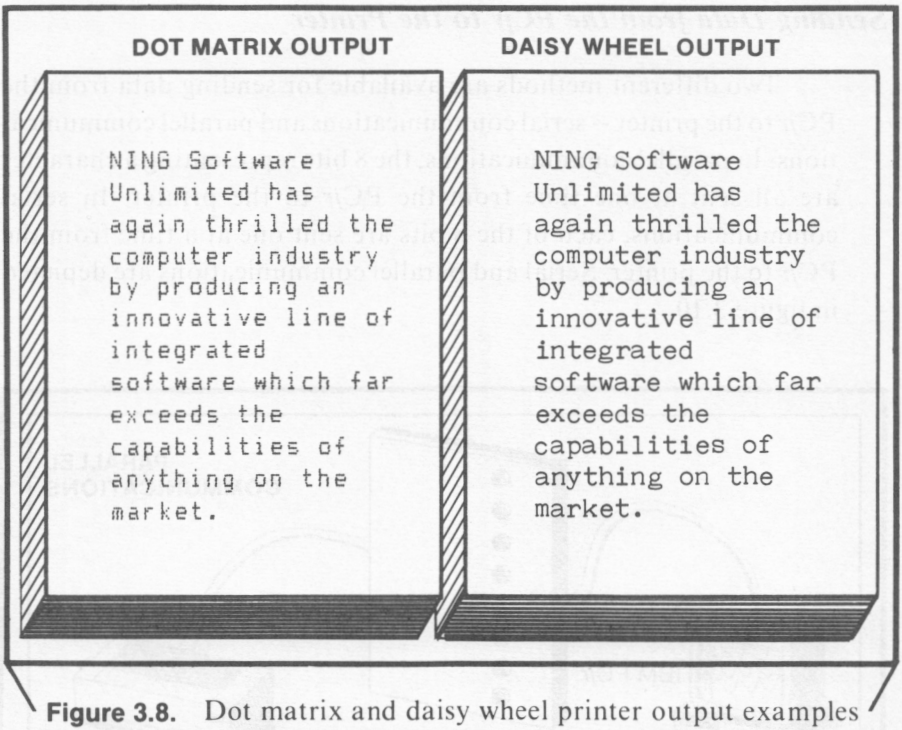


Figure 3.8. Dot matrix and daisy wheel printer output examples

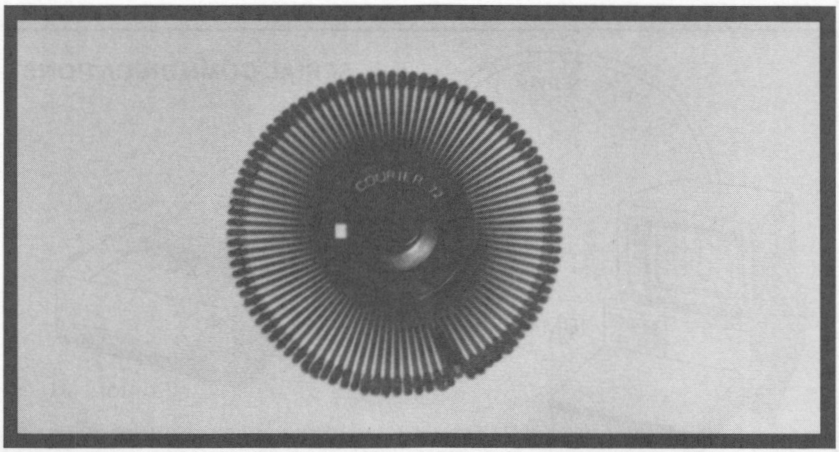


Figure 3.9. Daisy wheel

Sending Data from the PCjr to the Printer

Two different methods are available for sending data from the PC*jr* to the printer -- serial communications and parallel communications. In parallel communications, the 8 bits representing a character are all sent at one time from the PC*jr* to the printer. In serial communications, each of the 8 bits are sent one at a time from the PC*jr* to the printer. Serial and parallel communications are depicted in figure 3.10.

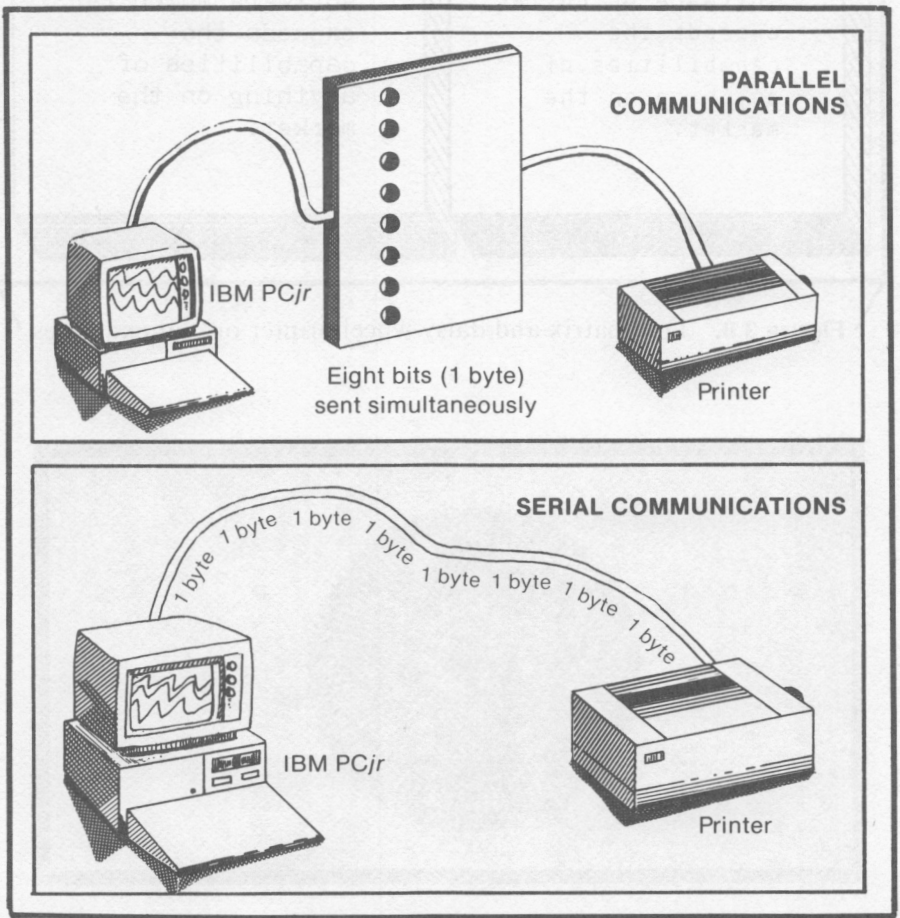


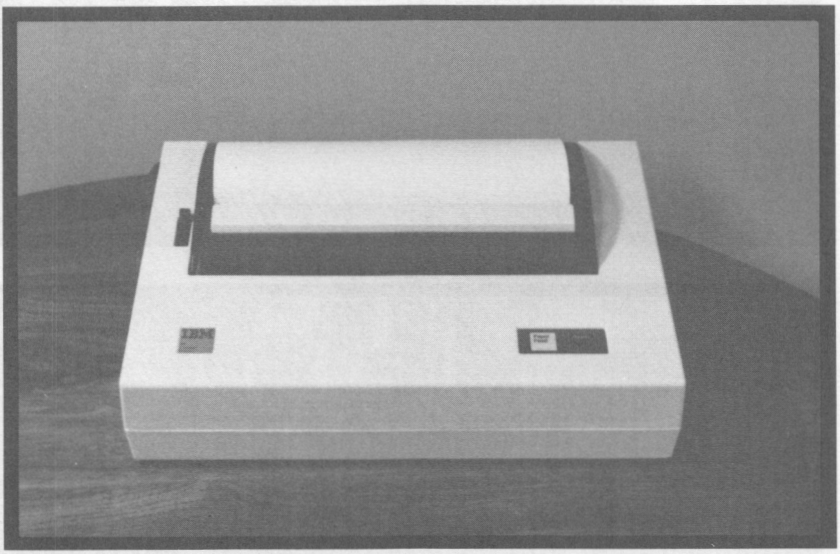
Figure 3.10. Serial and parallel communications

IBM Printers

IBM offers two printers that can be used with the PC*jr* -- the IBM Compact Printer and the IBM Graphics Printer. A number of printers manufactured by firms other than IBM can also be used with the PC*jr*.

The IBM Compact Printer is pictured in figure 3.11. The Compact Printer is a **thermal** dot matrix printer. Generally, dot matrix printers are **impact** dot matrix printers. With impact printers, printing is accomplished by a print head striking the paper surface. Thermal dot matrix printers utilize a less expensive technology than impact dot matrix printers. Thermal dot matrix printing requires a special, coated paper. When the paper's coating is struck by the thermal printer's hot impact surface, the character will be formed.

The IBM Compact Printer is a serial device. Data is sent to it from the PC*jr* during serial communications. The Serial Adapter Cable is required to install the Compact Printer.



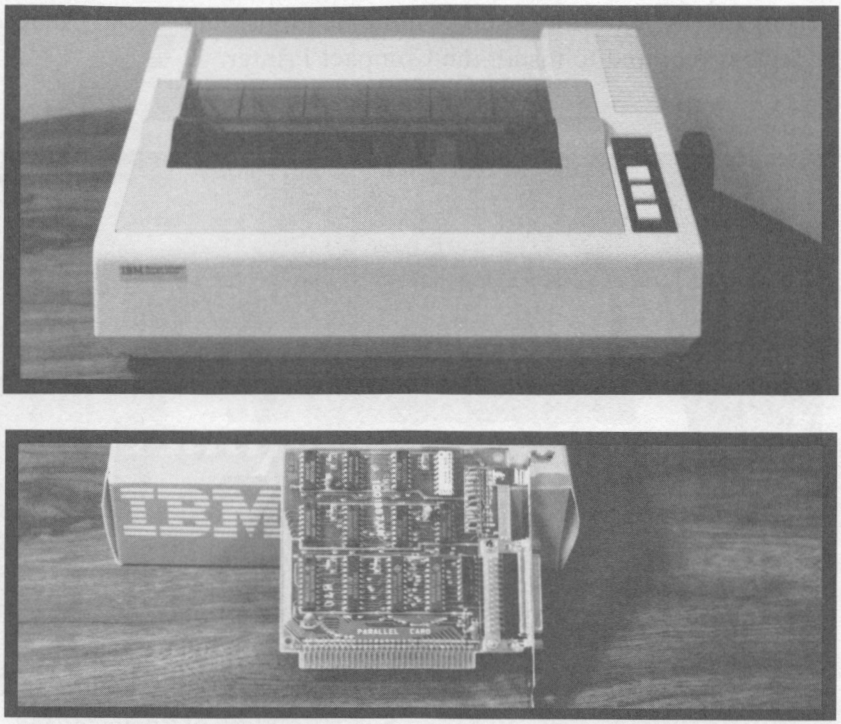
"Courtesy of Computerland"

Figure 3.11. IBM Compact Printer

The IBM Graphics Printer can also be used with the PC*jr*. This printer is shown in figure 3.12. Unlike the IBM Compact Printer, the IBM Graphics Printer is a parallel device. In order for the IBM Graphics Printer to be used with the PC*jr*, a device known as the Parallel Printer Attachment must be installed on the PC*jr*. This device will also allow the connection of many parallel printers not manufactured by IBM to the PC*jr*.

Joysticks

A joystick is a controller device used to position an object on the display. Generally, joysticks are used with game software. The IBM PC*jr* Attachable Joystick is pictured in figure 3.13.



"Courtesy of Computerland"

Figure 3.12. IBM Graphics Printer and Parallel Printer Adapter

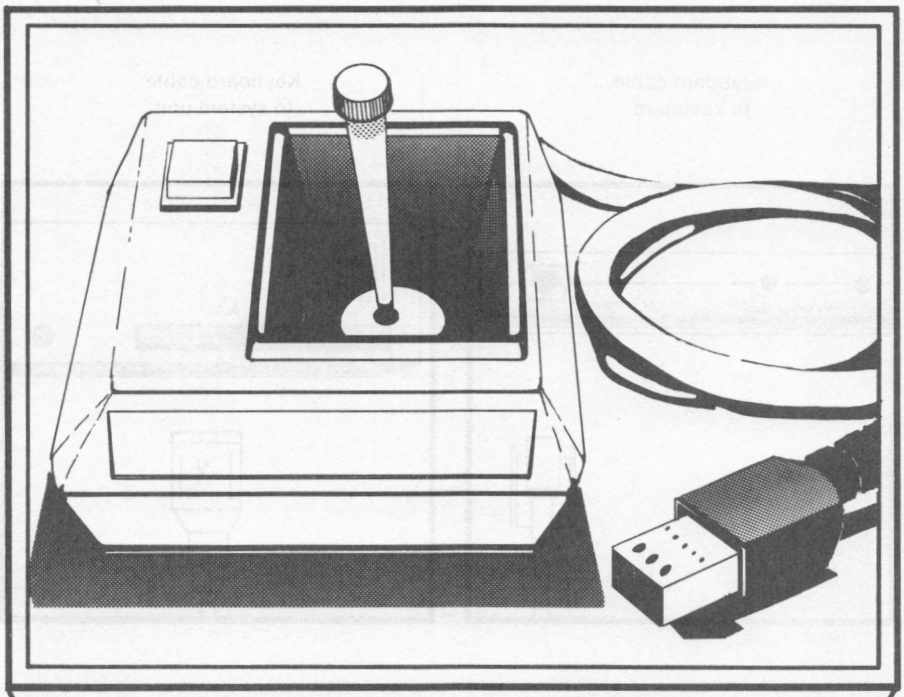


Figure 3.13. PC*jr* Attachable Joystick

PCjr Keyboard Cord

The PC*jr*'s keyboard can be connected to the system unit using the keyboard cord. Generally, a physical connection is not necessary as data is communicated from the keyboard to the system unit via an infrared link. The keyboard cord must be installed if more than one PC*jr* is being used in the same room. Otherwise, the infrared signals will become confused. The connection between the keyboard cable and the keyboard and system unit is depicted in figure 3.14.

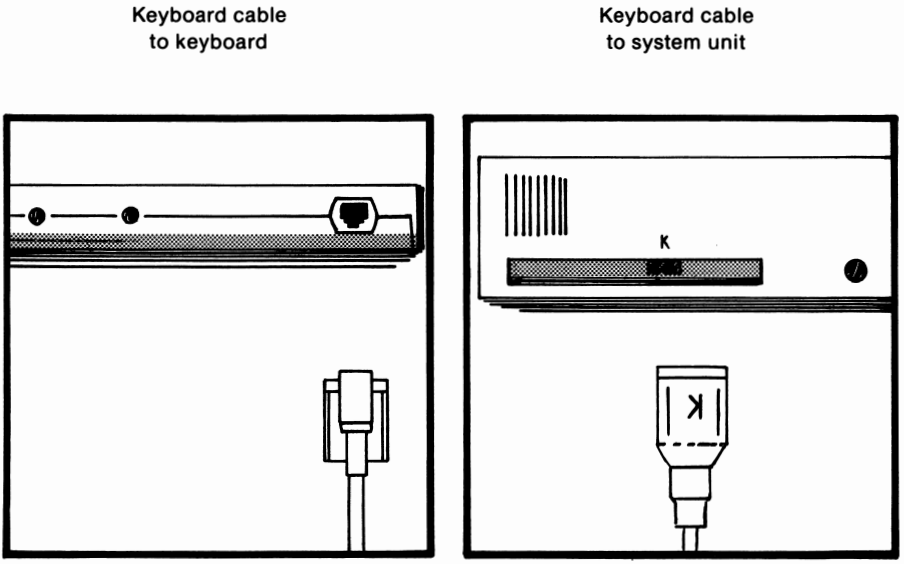


Figure 3.14. Keyboard cable connection

PCjr Memory and Display Expansion

The PC*jr*'s Memory and Display Expansion device is standard on the IBM PC*jr* enhanced model and optional on the entry model. This device is pictured in figure 3.15. This device expands the PC*jr*'s RAM memory capacity from 64K to 128K. This allows the PC*jr* to display 80 characters per line of video output rather than 40. The Memory and Display Expansion device is installed on the PC*jr*'s system board.

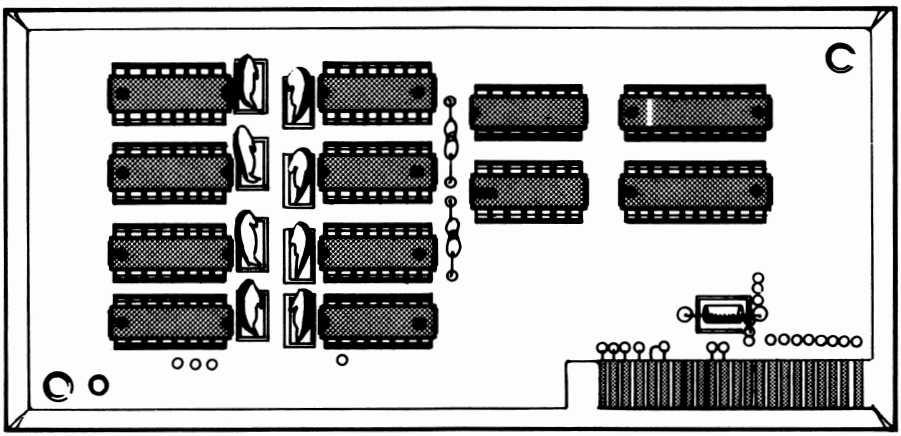


Figure 3.15. PC*jr* Memory and Display Expansion

PCjr Internal Modem

A modem allows communications between one computer and another located some distance away via telephone lines. A modem translates the sending computer's data into tones which can be sent over telephone lines. A modem also decodes tones received from the sending computer into data which can be accepted by the PC*jr*. The Internal Modem used with the PC*jr* is installed in the system unit. This device is pictured in figure 3.16.

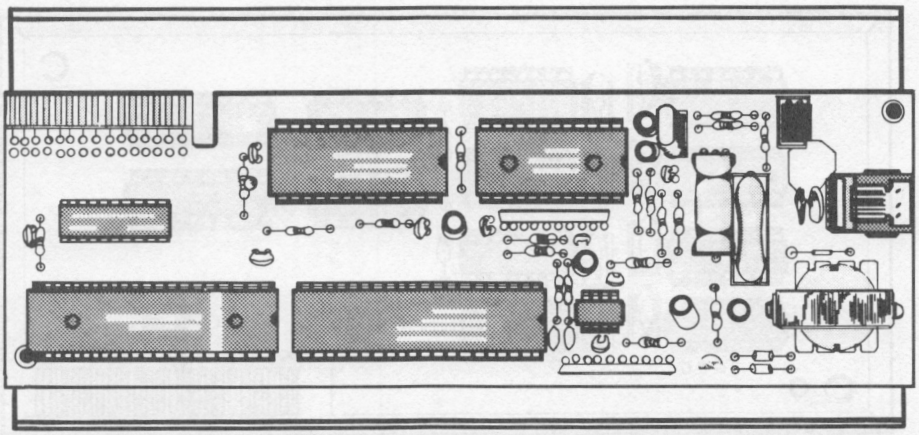


Figure 3.16. PC*jr* Internal Modem

Cassette Recorder/Player

The IBM PC*jr* can utilize a cassette recorder rather than a diskette drive for program and data storage. Cassette recorders are the most inexpensive data storage devices available for home computers. They provide a low cost and reliable means of data storage for the budget-minded home computer consumer. However, they offer only sequential access. Random access is not possible using a cassette recorder.

A cassette recorder uses standard cassette tapes to store data. It is good practice to use only high quality cassette tapes to save programs and data. Using lesser quality cassette tapes could result in the loss of programs and data.

The PC*jr* can use nearly any type of cassette recorder. However, a special adapter cable must be purchased in order to connect the cassette recorder to the PC*jr*. The PC*jr* adapter cable for cassette is pictured in figure 3.17.

PCjr Cartridges

Cartridges are often used to store PC*jr* programs. A PC*jr* cartridge consists of 32K of ROM enclosed in a plastic case. A PC*jr* cartridge is pictured in figure 3.18. PC*jr* cartridges should be inserted in one of the PC*jr*'s two cartridge slots as shown in figure 3.19.

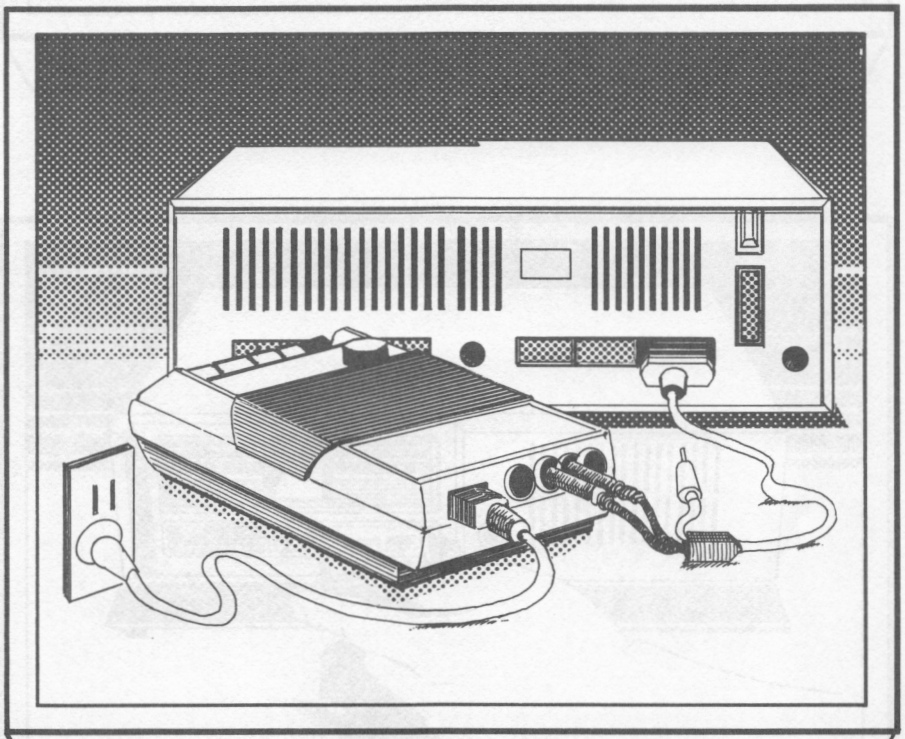


Figure 3.17. Cassette recorder/PC*jr* installation

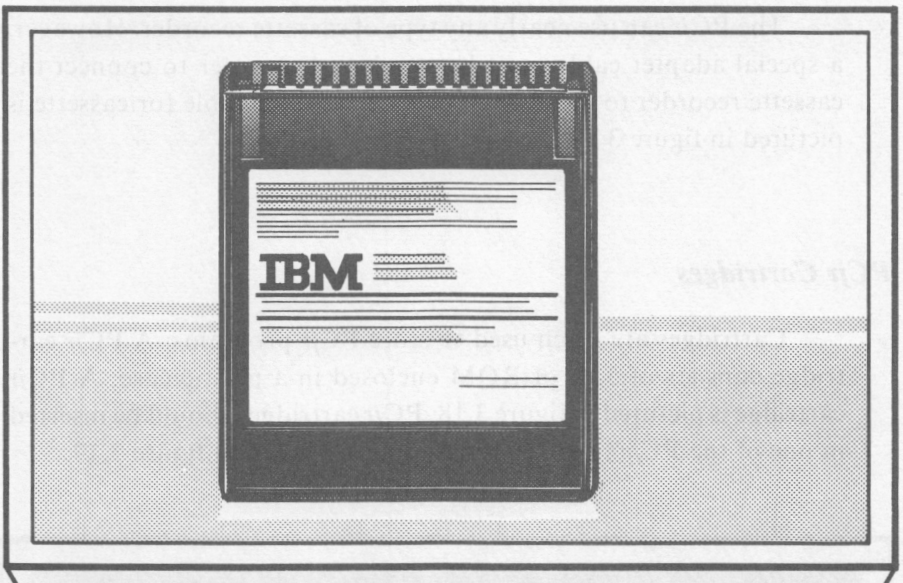


Figure 3.18. PC*jr* cartridge

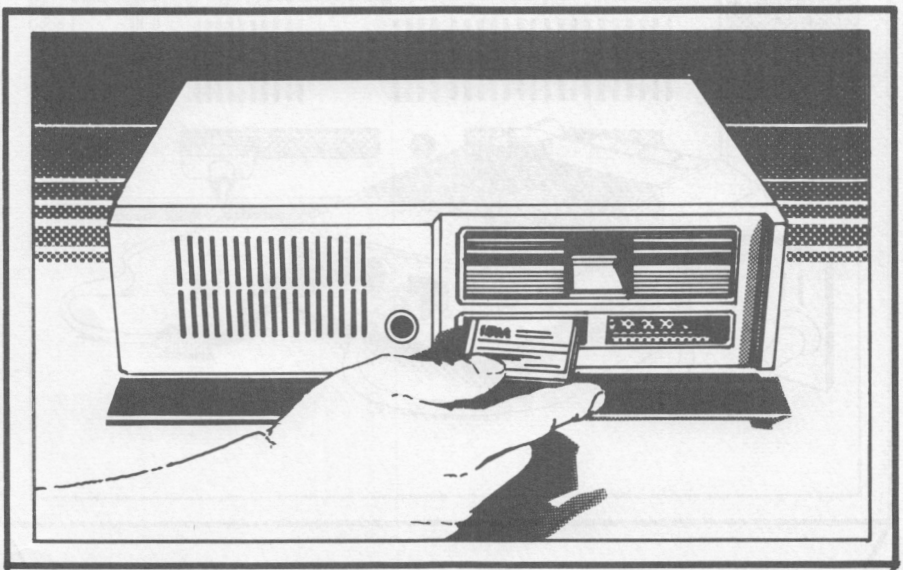
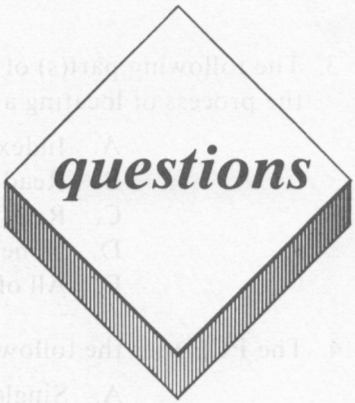


Figure 3.19. Inserting a PC*jr* cartridge



True or False

1. Random access is not possible with data stored on a diskette.
2. A track is a concentric band of data on the diskette surface.
3. Dot matrix printer output is of superior quality to daisy wheel printer output.
4. A modem is used to send data from the PC*jr* to a cassette recorder.
5. Single density diskettes can store more data than double density diskettes.

Multiple Choice

1. The following devices are example of peripherals:

   A. RAM
   B. IBM PC*jr*
   C. IBM Graphics Printer
   D. ROM
   E. All of the above

2. How many bytes can be stored on an individual sector on a diskette used with the PC*jr*?

   A. 256
   B. 64
   C. 10
   D. 512
   E. None of the above

3. The following part(s) of a diskette and disk drive are essential in the process of locating an individual sector:

   A. Index hole
   B. Read/write head
   C. Read/write head slot
   D. None of the above
   E. All of the above

4. The PC*jr* uses the following type of diskette:

   A. Single sided; single density
   B. Single sided; double density
   C. Double sided; single density
   D. Double sided; double density
   E. None of the above

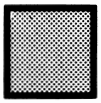
5. In serial communications, the following number of bits are sent simultaneously to the receiving device:

   A. 1
   B. 4
   C. 8
   D. 2
   E. None of the above

Essay

1. Describe parallel and serial communications.
2. Define sequential and random access.
3. Describe the process whereby DOS divides a diskette's surface so that data can be more easily accessed.

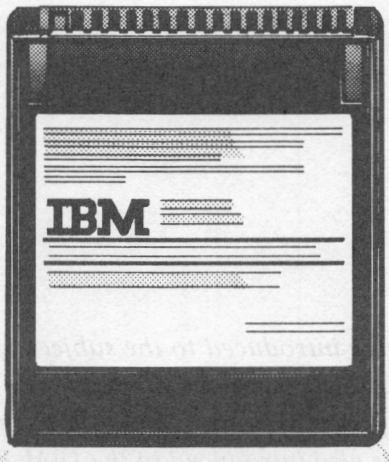
Section 2



In the first section, you were introduced to the subject of computing in general. You gained a general understanding of how a computer functions, and you were introduced to a number of terms used in computing. You were also introduced to the IBM PCjr as well as its peripherals and add-on devices. The first section should have provided you with the basic background you will need to continue your journey towards computer literacy.

Our next section is entitled, "DISCOVERY". In this section, you will begin learning about computers by actually operating and programming your PCjr. We will attempt to accomplish the following goals in this section:

- ☐ *Gain an understanding of the background of the Microsoft BASIC language*
- ☐ *Learn how to start-up the PCjr*
- ☐ *Learn how to enter and run a simple BASIC program*
- ☐ *Gain an understanding of how the PCjr processes a simple program*
- ☐ *Learn how to list a BASIC program on the video display*
- ☐ *Learn how to generate program lines automatically*
- ☐ *Learn how to renumber the lines in a BASIC program*
- ☐ *Learn how to edit a BASIC program*



Introduction to BASIC

lesson 4

Lesson Goals

- *Gain an understanding of the background and evolution of Microsoft BASIC*
- *Define the different classifications of software and languages*
- *Gain an understanding of the relationship between Cassette BASIC and Cartridge BASIC*

Introduction

In this lesson, we will provide background information on the BASIC programming language. BASIC is a simple, easy-to-use programming language that is ideal for the first-time programmer. Once you have learned how to program in BASIC, you will find it relatively easy to learn how to program in other languages such as FORTRAN, PASCAL, and COBOL.

Programming Languages High Level, Machine, and Assembly

A **program** can be defined as a set of instructions arranged in a specific sequence which directs the operation of a digital computer. A **language** can be defined as a set of words or symbols that can be used for communicating. A **programming language** can be defined as a set of words or symbols that can be used to communicate instructions to a digital computer. The BASIC programming language is included as a standard feature with the PC*jr*.

BASIC is a **high level** programming language. A high level language does not require that the programmer have an understanding of the internal workings of the computer. With a **machine** or an **assembly** language, the programmer must have an in-depth understanding of the computer and its microprocessor in order to write programs.

Numbers are used in machine language programs to send instructions to the microprocessor. For example, the hexadecimal number EA would instruct the PC*jr* to jump to another memory location and resume execution at that location. **Mnemonics** are used in assembly language programs to communicate instructions. The mnemonic JMP is used in Intel 8088 assembly language to instruct the PC*jr* to jump to another memory location and resume execution there.

With a high level language such as BASIC, commands are generally specified in English words that can be associated with the operation to be performed. For example, the BASIC command

PRINT instructs the computer to display information. It is generally much easier to write a program in a high level language than in a machine or assembly language.

History of BASIC

The BASIC language was originally developed in the early 1960's by John G. Kemeny and Thomas E. Kurtz of Dartmouth College. Over the years, a number of different versions of the BASIC language have been developed. The version of BASIC used on the IBM PC*jr* is Microsoft BASIC, which is marketed by Microsoft Corporation. Microsoft BASIC is used on a number of different personal computers.

Microsoft BASIC was originally developed in 1975 for the MITS Altair computer by William Gates and Paul Allen. Gates and Allen eventually formed their own firm, Microsoft Corporation, to market their version of BASIC.

Microsoft BASIC became the industry standard in the personal computer field, and Microsoft Corporation went on to develop and/or market a number of other personal computer related items, including the disk operating system used on the IBM PC*jr*.

Compiled vs. Interpreted Languages

Computer languages are often distinguished as being either **compiled** or **interpreted** languages. Microsoft BASIC is an interpreted language.

A compiled language program consists of the **source code** and the **compiled code.** The source code consists of the program statements in their original form. For example, the following is a line of source code from a program written in the CBASIC compiled language:

```
100 INPUT "ENTER TODAY'S DATE:";DATE.1
```

The source code is processed by a program known as a **compiler** into the compiled code. The compiled code is very similar to the machine language used by the microprocessor. The compiled code is the code actually used when a compiled program is run. A program known as a **run-time monitor** is used to run the compiled program.

An interpreted language consists of only the source code. The source code is translated line-by-line directly into machine language instructions. One advantage of interpreted languages over compiled languages is that interpreted language programs are more easily developed. When working with interpreted languages, a programmer need only write a program, enter it, run it, and alter it at his leisure. When working with a compiled language, the source code must be recompiled every time it is edited. This can be frustrating during the program debugging process.

One advantage of compiled languages over interpreted languages is that execution time is much faster. The compiled code is much closer to the machine language than the source code. Since interpretation is not necessary, execution of compiled code is much faster.

Cassette and Cartridge BASIC

The Microsoft BASIC interpreter is supplied with the PC*jr* as two different parts. The Cassette BASIC portion of the interpreter is contained in 32K of ROM on the PC*jr*'s system board. Cassette BASIC includes the majority of Microsoft BASIC's many features. Although Cassette BASIC allows for the storage of data on a cassette player/recorder, it cannot be used when data is being stored on diskette.

Cartridge BASIC is contained on a PC*jr* cartridge. When this cartridge is inserted into either of the PC*jr*'s two cartridge slots, Cartridge BASIC will be active in addition to Cassette BASIC. Cartridge BASIC allows data to be stored on diskette as well as cassette. Cartridge BASIC also includes a number of BASIC commands not available in Cassette BASIC. These include:

CIRCLE	WINDOW
PUT	PALETTE
GET	PALETTE USING
PAINT	PLAY
DRAW	TERM
VIEW	

Cartridge BASIC also allows for additional graphics screen modes beyond those found in Cassette BASIC.

Types of Software

At the beginning of this chapter, we defined a program as a set of instructions which direct the computer's operation. The term **software** is almost synonymous with programs. Software can be defined as the various programs that control the computer's operation. We have already briefly examined two examples of PC*jr* software, the Microsoft BASIC interpreter and the DOS disk operating system.

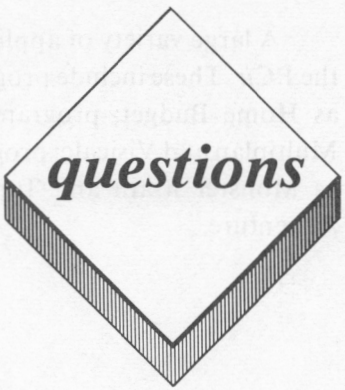
Software can be divided among three general classifications:

- ☐ Operating system software
- ☐ Language software
- ☐ Applications software

The PC*jr*'s DOS is an example of operating system software, and Microsoft BASIC is an example of language software. Applications software can be defined as programs designed to accomplish a specific task that is of some value to the user. Examples of applications programs include games, word processing programs, spreadsheets, and database systems.

Generally, applications programs are stored on cassette or diskette and are transferred into RAM, where the program is available to the computer. Applications programs can also be stored in a permanent form on a ROM cartridge. This ROM cartridge can be plugged into one of the PC*jr*'s cartridge slots.

A large variety of applications software is available for use with the PC*jr*. These include programs which can be used in the home such as Home Budget; programs which can be used at work such as Multiplan and Visicalc; programs with educational applications such as Monster Math and Turtle Power; and finally, games such as Adventure.



True or False

1. Mnemonics are used to communicate instructions in a high level language.
2. Machine language programming requires an in-depth understanding of the workings of the computer and its microprocessor.
3. An interpreted language program consists only of source code.
4. Cartridge BASIC is contained in 32K of ROM on the PC*jr*'s system board.
5. The PC*jr*'s Microsoft BASIC interpreter is an example of applications software.

Multiple Choice

1. The following are examples of high level programming languages:
   A. Microsoft BASIC
   B. FORTRAN
   C. COBOL
   D. PASCAL
   E. All of the above

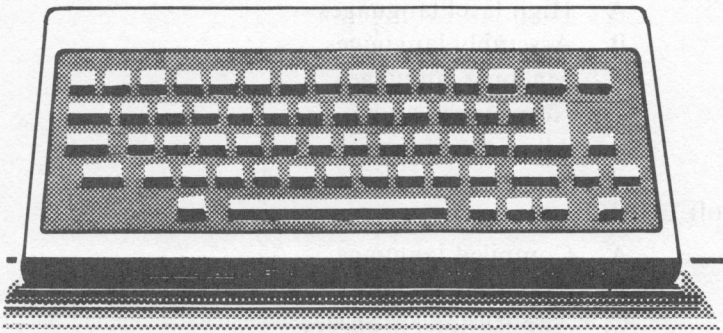
2. Numbers are used to communicate instructions in the following:
   A. High level languages
   B. Assembly languages
   C. Machine languages
   D. None of the above
   E. All of the above

3. Microsoft BASIC is a:
   A. Compiled language
   B. Operating system
   C. Applications program
   D. Interpreted language
   E. None of the above

4. One advantage of an interpreted language over a compiled language is:
   A. Programs can be written more easily
   B. Programs can be more easily debugged
   C. The compilation step is not necessary
   D. None of the above
   E. All of the above

Essay

1. Define the terms program, language, and programming language.
2. Define the terms machine language, assembly language, and high level language.
3. What is the difference between a compiled and an interpreted language?
4. What are the relative advantages and disadvantages of compiled and interpreted languages?



Getting Started with BASIC

lesson 5

Lesson Goals

- ☐ *Learn how to start-up the PCjr*
- ☐ *Gain an understanding of the PCjr's keyboard*
- ☐ *Learn how to send commands to the PCjr*

Introduction

In this lesson, we will begin learning about computing via the hands-on approach. By hands-on, we mean that we will be actually using the computer while we are learning about it. We will learn how to start-up the PC*jr* as well as how to use its keyboard to input information. By the time you have progressed to the end of this lesson, you will have learned how to input a simple command to which the PC*jr* will respond.

PCjr System Start-Up Review

Before actually starting up the PC*jr*, let's make sure that the system is set up properly. A typical PC*jr* system set-up is depicted in figures 5.1 and 5.2.

Notice that a minimum of four components are required for a functional system. These include:

- ☐ System unit
- ☐ Keyboard
- ☐ Video display
- ☐ Power supply/transformer

Note that a physical connection is not required between the keyboard and the system unit. The keyboard can communicate with the system unit using the infrared link. From figure 5.2, we can see that a physical link is required between the video display and the system unit. The power supply/transformer also plugs into the rear of the system unit.

BASIC Start-Up

You can start-up the PC*jr* either with the diskette drive or without the diskette drive. We will discuss both methods in the following two sections.

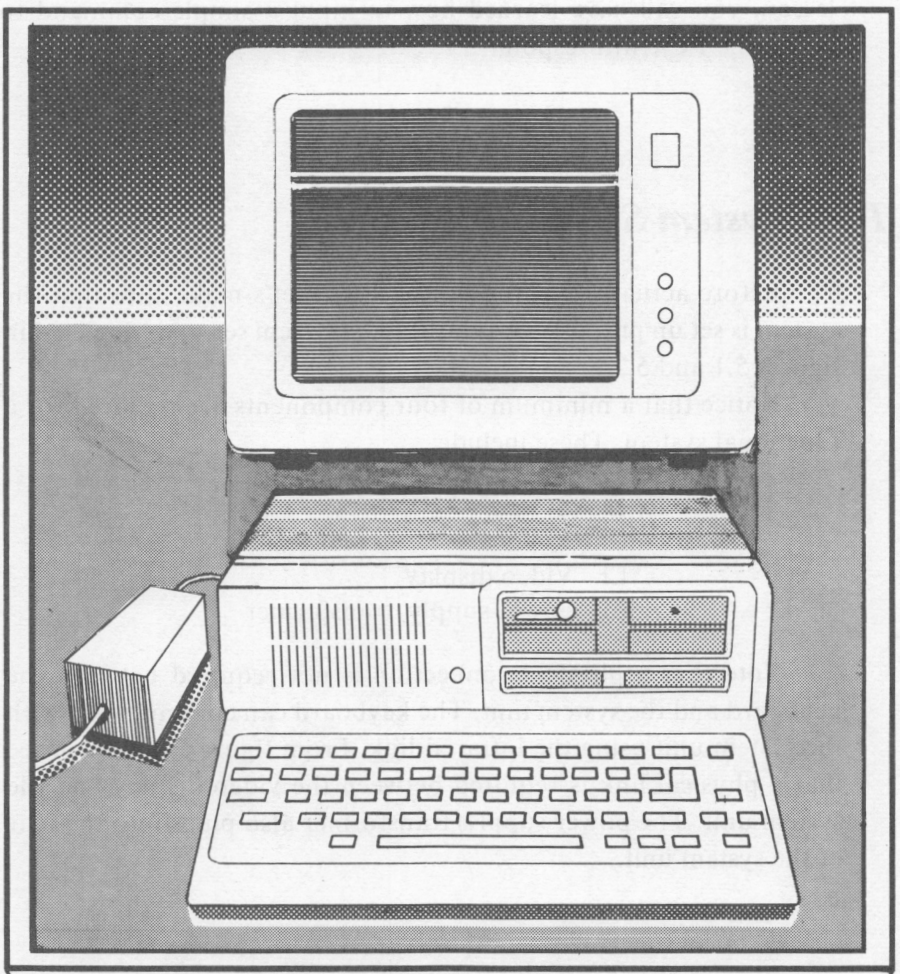


Figure 5.1. PC*jr* system set-up (front view)

PCjr Start-Up Without the Diskette Drive

In this section, we will outline the steps that should be taken to start-up your PC*jr* without using the diskette drive. If the PC*jr* does not appear to be functioning properly during the start-up procedure, merely turn the PC*jr*'s power off and begin over again.

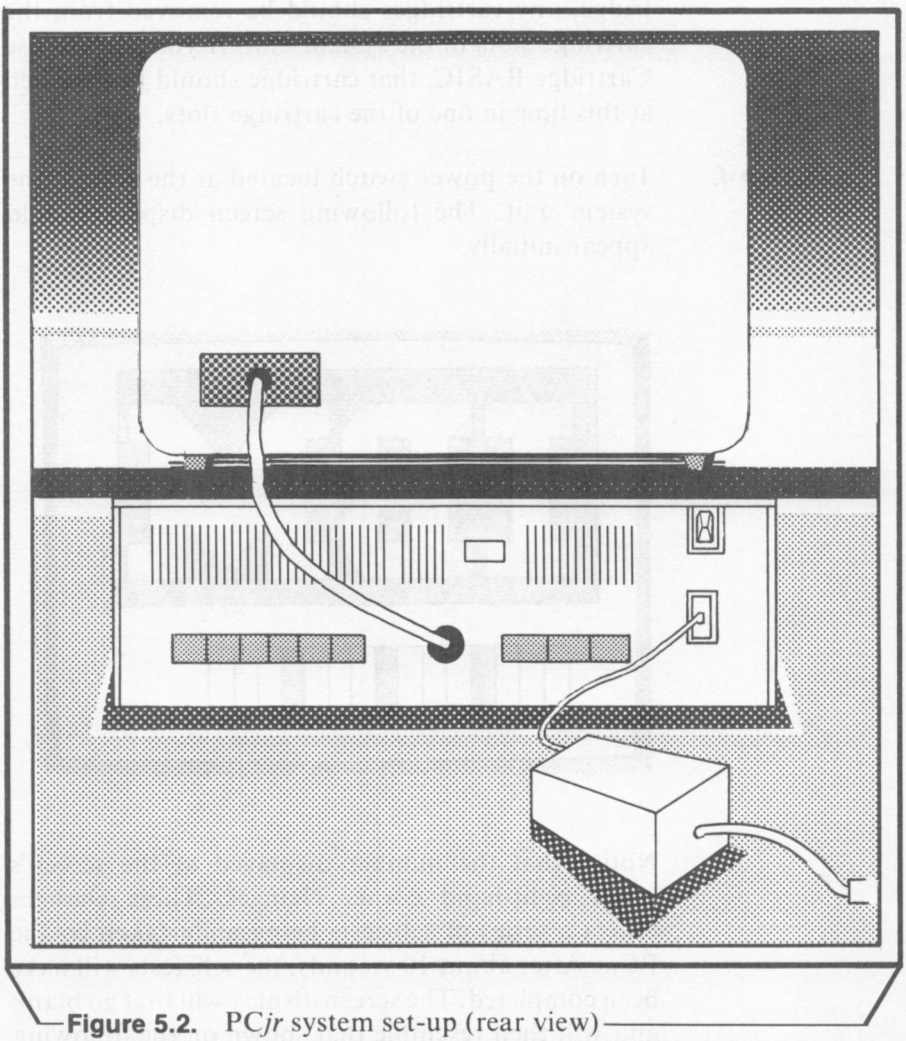


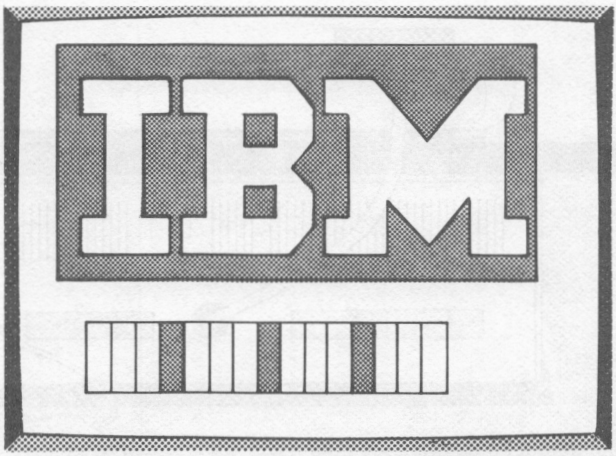
Figure 5.2. PC*jr* system set-up (rear view)

Step 1. If the PC*jr*'s power is on, turn it off. The power switch is located at the rear of the system unit as shown in figure 5.2.

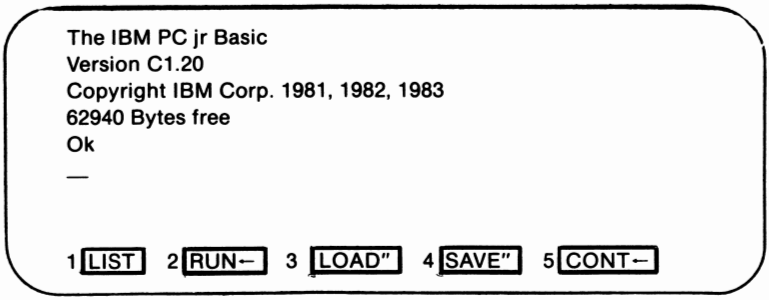
Step 2. The video display device should be turned on. The volume control knob should be turned down.

Step 3. With the exception of the Cartridge BASIC cartridge, any cartridges should be removed from the cartridge slots in the system unit. If you wish to use Cartridge BASIC, that cartridge should be installed at this time in one of the cartridge slots.

Step 4. Turn on the power switch located at the rear of the system unit. The following screen display should appear initially:



Notice that the number displayed at the screen's lower right-hand corner changes. These changes reflect a series of self-tests being undertaken by the PC*jr*. After about 10 seconds, the self-tests will have been completed. The screen display will first go blank and will then resemble that shown on the following page:



Cassette BASIC is known as "Version C". If the BASIC cartridge had been inserted, "Version J" would have replaced "Version C" at the beginning of line two. Cartridge BASIC is known as "Version J".

PCjr Start-Up with the Diskette Drive

If your PC*jr* does not have a disk drive, then you can ignore this section. Use the instructions in the preceding section to start-up your PC*jr*. If your PC*jr* does have a diskette drive, then use the steps outlined in this section for start-up.

Cartridge BASIC is required for disk drive usage. Therefore, the Cartridge BASIC cartridge should be inserted in one of the PC*jr*'s cartridge slots if the diskette drive start-up procedure is to be undertaken. If you do not have Cartridge BASIC, use the start-up procedure for a PC*jr* without a diskette drive.

The DOS diskette is also required. This diskette is supplied with PC*jr*'s that include a diskette drive.

The steps involved in starting up the PC*jr* with a diskette drive are given below:

Step 1. If the PC*jr* is not already turned off, do so now.

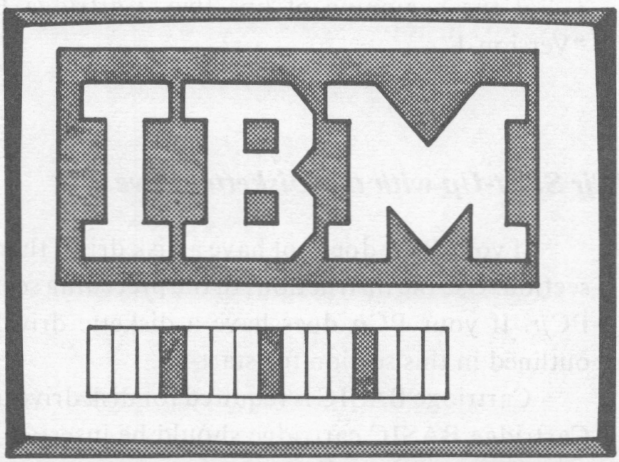
Step 2. The video display should be turned on. The volume

control knob should be turned down.

Step 3. Insert the DOS diskette in the diskette drive. Follow the instructions outlined on page 64.

Step 4. With the exception of the Cartridge BASIC cartridge, any cartridges should be removed from the cartridge slots in the system unit. As mentioned earlier, Cartridge BASIC must be installed in order for you to use the diskette drive.

Step 5. Turn on the power switch located at the rear of the system unit. The following display will appear:



The number displayed on the screen's lower right-hand corner will change as the PC*jr* performs a series of self-tests. After about 10 seconds, the self-tests will have been completed. The display screen will first go blank. Then, a beep will be output by the PC*jr*, and the red lamp on the front of the diskette drive will light. After a few seconds, the following will be displayed on the screen:

```
Current date is Tues 1-01-1980
Enter new date:
```

At this point, you can either enter a date using the keyboard, or you can press the Enter key. The date entered is known as the system date. The system date is used by the PC*jr*'s DOS in a number of different file handling operations. If the Enter key is pressed, the system date will default to the date displayed on the screen, January 1, 1980. If a new date is entered using the keyboard, that date will be used as the system date.

A certain format must be used when entering the system date. If this format is not used, the following error message will be displayed:

```
Invalid date
```

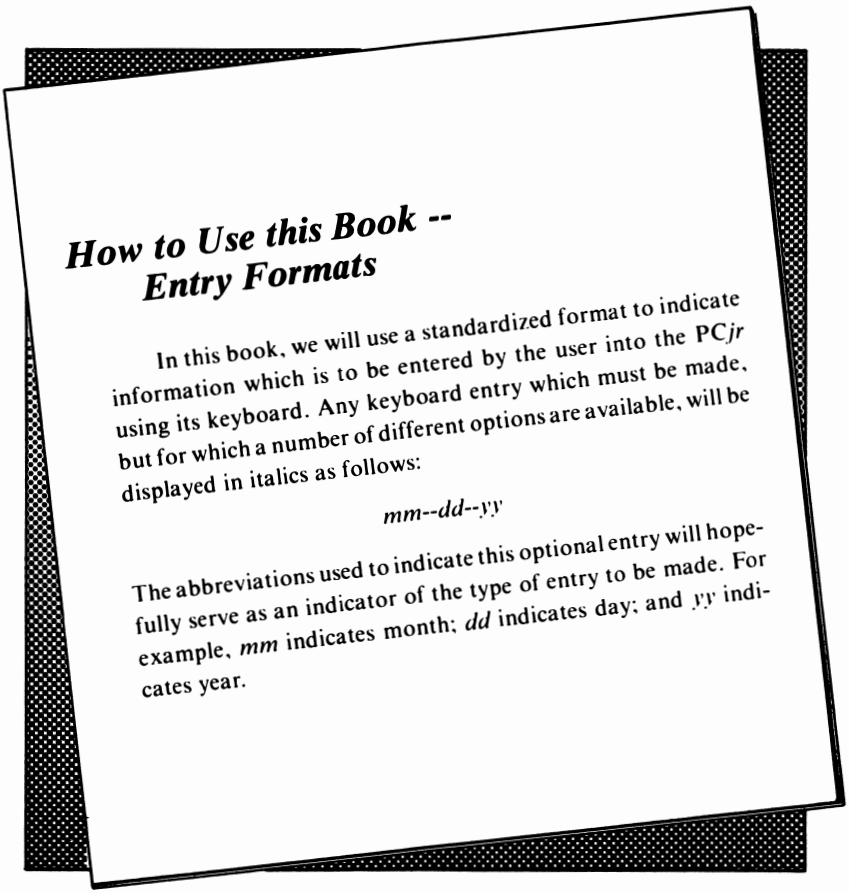
This entry format is as follows:

mm--dd--yy
or
mm/dd/yy

mm indicates the month. It must consist of one or two integers in the range 1 to 12. *dd* indicates the day. It must be one or two integers in the range from 1 to 31. *yy* indicates the year. It must be two digits in the range from 80 to 99.

The following are examples of valid and invalid system date entries:

Valid System Date Entries	01/04/84	5-1-84
	3/21/85	7-30-82
Invalid System Date Entries	2.12.84	6/84
	January 12, 1982	2-31-84



How to Use this Book -- Entry Formats

In this book, we will use a standardized format to indicate information which is to be entered by the user into the PC*jr* using its keyboard. Any keyboard entry which must be made, but for which a number of different options are available, will be displayed in italics as follows:

mm--dd--yy

The abbreviations used to indicate this optional entry will hopefully serve as an indicator of the type of entry to be made. For example, *mm* indicates month; *dd* indicates day; and *yy* indicates year.

Step 6. Once the system date has been entered, the following screen display will appear:

```
Current date is Tues 1-01-1980
Enter new date:
Current time is 0:00:34.49
Enter new time:
```

The PC*jr* is requesting entry of the system time. Again, you can either enter a date via the keyboard, or press the Enter key. Pressing the Enter key causes the system time to default to that displayed on the screen.

The following entry format must be observed if the system time is entered using the keyboard:

hh:mm:ss.xx

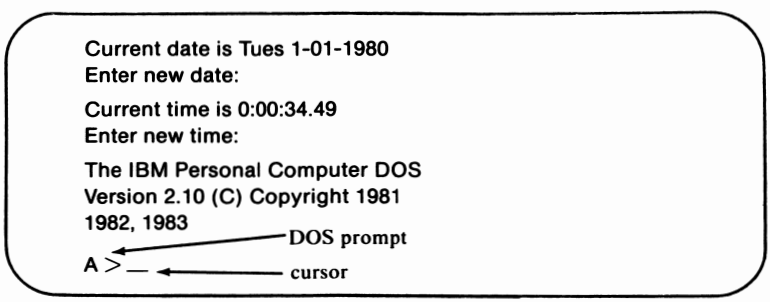
hh indicates the hour of the day. One or two digits should be used in the range 0 to 23. Hours in the PM will be assigned integers greater than 12. For instance, 3 PM would be assigned the integer 15. *mm* indicates minutes. This can be entered as one or two digits in the range 0 to 59. *ss* indicates seconds. This can be entered as one or two digits in the range 0 to 59. *xx* indicates hundredths of seconds. This can be entered as one or two digits in the range 0 to 99. The following entries are optional:

mm *ss* *xx*

The following are all valid entries for the system time:

1	14:12:7
23:01	8:1:1.9

Step 7. After the system time has been entered, the screen display will appear as follows:



At this point, the PC*jr* is ready to accept DOS commands. However, it is not ready to accept BASIC commands. A > is known as the DOS prompt. These characters serve as an indication that a DOS command can be entered.

Notice the flashing symbol to the right of the DOS prompt. This symbol is known as the cursor. The cursor indicates that the computer is ready to accept keyboard entries.

Since we are interested in using Microsoft BASIC, our next step will be to activate the Microsoft BASIC interpreter. We can do so by entering the following characters:

BASIC ↵

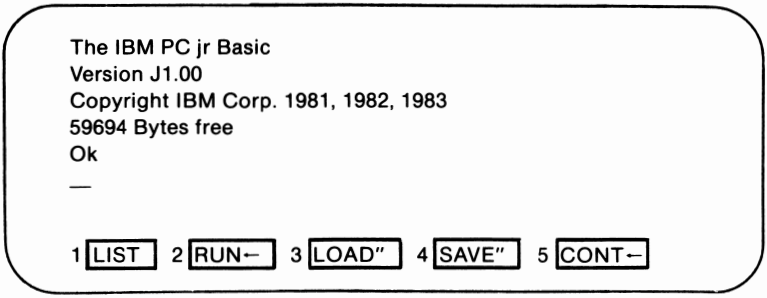
BASIC can be entered either as upper or lower case letters. The symbol ↵ will be used in this book as an indication that the Enter key is to be pressed.

If you make a typing error, the PC*jr* will probably display a message as shown below:

```
Current date is Tues 1-01-1980
Enter new date:
Current time is 0:00:34.49
Enter new time:
The IBM Personal Computer DOS
Version 2.10 (C) Copyright 1981,
1982, 1983
A > VASIC
Bad command or filename
A >_
```

If you do make a typing mistake, don't worry. Just reenter the correct characters.

Once the correct entry has been made, the screen will go blank for a second or two, and the screen will then appear as follows:



Notice the Ok. This is the BASIC prompt. Ok indicates that a BASIC command can be entered.

Adjusting the Screen

Upon start-up, you may find it necessary to adjust the PC*jr*'s screen. This is a relatively simple matter. First, hold down the Alt key, which is located to the immediate left of the space bar. While holding the Alt key, also hold the Ctrl key. This key is located along the keyboard's left-hand side. While holding both of these keys, press the → key located on the keyboard's right-hand side.

Notice that every time the → key is pressed, the screen display moves to the right. If the ← key is pressed with Ctrl and Alt, the screen display will move to the left.

Warm Boot

The procedure for starting up a computer when it is powered off is known as a cold boot (or start). The procedure for restarting a computer when its power is on is known as a warm boot.

You will often encounter situations where an incorrect entry or error condition will make it easier to just restart BASIC and erase the contents in memory, rather than to attempt to correct the situation. Such situations can be corrected by performing a warm boot.

To perform a warm boot, press the Alt and Ctrl keys simultaneously. While you are holding these keys down, press the Del key, and then release all three keys. The computer will react much as if it had been started with the power off.

Whenever you perform a warm boot, remember that any existing data in the PC*jr*'s memory will be erased.

PCjr Keyboard

The IBM PC*jr*'s keyboard is shown in figure 5.3. Most of its keys correspond to those found on a standard typewriter. Notice that many of the keys are marked with two characters, one of which is in white and the other in black, green, or blue.

When one character is displayed in white and the other character in black, the character displayed in white will be generated when that key is pressed. The character in black can be accessed by holding down the Shift key and then pressing the key with the desired character.

Even though the keys corresponding to the letters of the alphabet are indicated on the keyboard with only one character, they also can actually indicate two. When a letter key is pressed without the Shift key being depressed, the lowercase letter will be generated. When Shift is pressed, the uppercase letter will be generated.

For example, press the A key. The lowercase "a" will be displayed on the screen. Now, press the Shift key and hold it while pressing the A key. Notice that the uppercase "A" will be displayed.

Obviously, it would be difficult to enter a large number of uppercase letters using the Shift key, as one hand would be constantly busy holding the Shift key. This problem can be solved by using the PC*jr*'s CapsLock key.

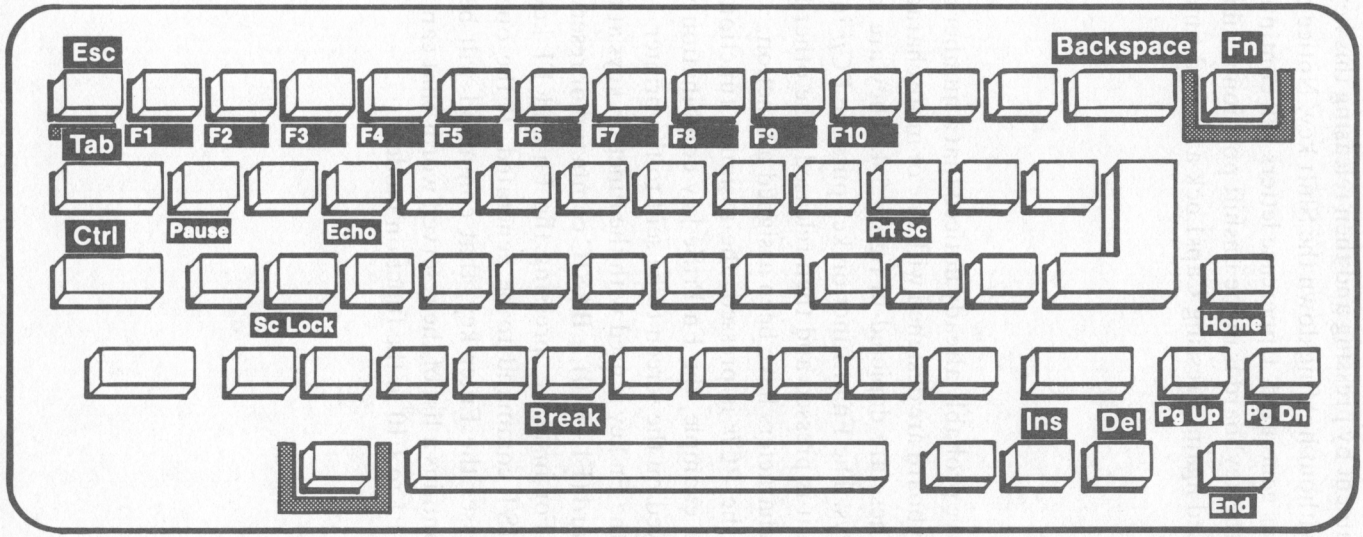


Figure 5.3. PC*jr* keyboard

Experiment by pressing and then releasing this key. Then, press the A key without holding down the Shift key. Notice that an uppercase "A" was generated. Uppercase letters will continue to be generated when the keyboard is in the unshift position until the CapsLock key is pressed again. Pressing CapsLock a second time will turn off this feature.

Fn Key

You have probably already noticed that a number of the keys on the PC*jr* keyboard are designed with one or more characters outlined in green. These are depicted in figure 5.4. They are known as the function keys. The Fn or function key puts the PC*jr* in the function mode when it is pressed and then released. The other keys denoted with green characters have been assigned a function.

When the PC*jr* is pressed, the assigned function will be performed. For example, the Fn PrtSc key combination will cause all data displayed on the screen to be sent to the printer.

When the Fn key is used with the function keys at the top of the PC*jr* keyboard (F1 - F10), a BASIC command or reserved word will be output. For instance, pressing the Fn and F1 keys will cause BASIC's LIST command to be generated. If the operator subsequently presses the Enter key, that command will be performed. Table 5.1 contains a list of the reserved words that can be generated by pressing F1 to F10 in the function mode.

Alt Key

The Alt and related keys are designated on the PC*jr*'s keyboard in blue. The Alt key places the PC*jr* keyboard in the alternate mode. The alternate mode is generally used to enter BASIC reserved words. By pressing and holding the Alt key with one of the letter keys, a BASIC reserved word will be generated.

For example, press the Alt and A keys. Notice that the BASIC keyword AUTO was generated. Table 5.1 includes a list of the BASIC reserved words that can be generated in the alternate mode.

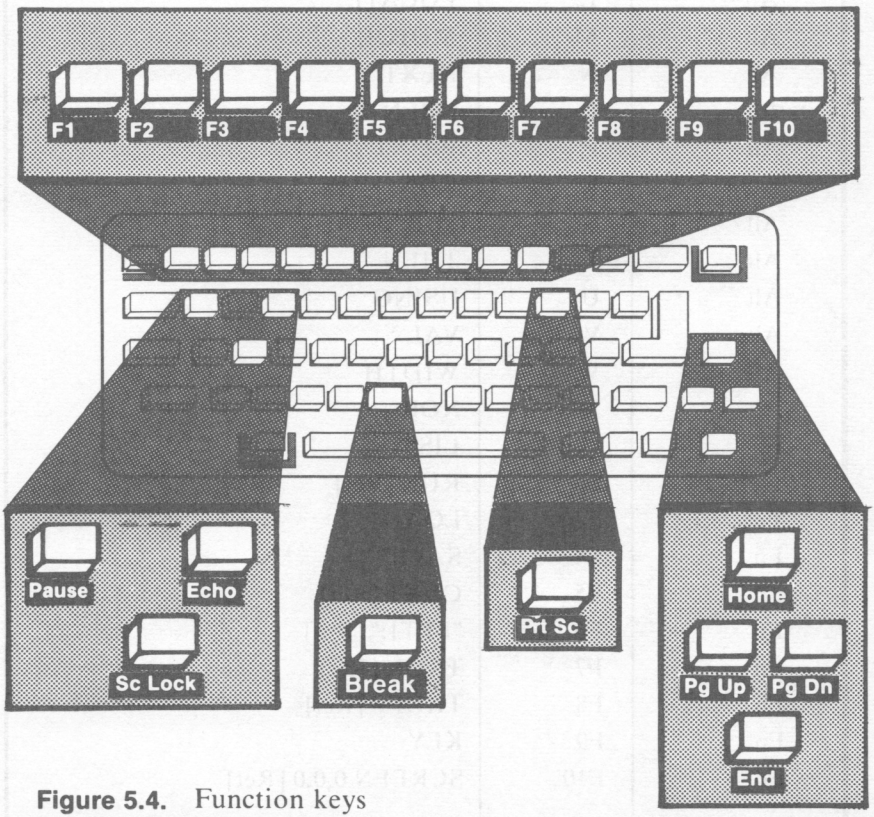


Figure 5.4. Function keys

Table 5.1 Fn and Alt key combinations

Hold Down	Press	Result
Alt	A	AUTO
Alt	B	BSAVE
Alt	C	COLOR
Alt	D	DELETE
Alt	E	ELSE
Alt	F	FOR
Alt	G	GOTO
Alt	H	HEX$
Alt	I	INPUT
Alt	K	KEY
Alt	L	LOCATE
Alt	M	MOTOR
Alt	N	NEXT
Alt	O	OPEN
Alt	P	PRINT
Alt	R	RUN
Alt	S	SCREEN
Alt	T	THEN
Alt	U	USING
Alt	V	VAL
Alt	W	WIDTH
Alt	X	XOR
Fn	F1	LIST
Fn	F2	RUN [Ret]
Fn	F3	LOAD"
Fn	F4	SAVE"
Fn	F5	CONT [Ret]
Fn	F6	"LPT1:" [Ret]
Fn	F7	TRON [Ret]
Fn	F8	TROFF [Ret]
Fn	F9	KEY
Fn	F10	SCREEN 0,0,0 [Ret]

Fn	Pause	Temporarily halts computer operation until a key (other than Shift) is pressed
Fn	Echo	When pressed once, causes text sent to the screen to be sent to the printer as well. When pressed again, text will only be sent to the screen
Fn	PrtSc	Results in data displayed on screen being output to the printer
Fn	Break	Stops execution of a BASIC program
Fn	ScLock + Break	Halts program execution indicating line number where execution stops
Fn	Home	Positions cursor to screen's upper right-hand corner

Table 5.1. (cont.) Fn and Alt key combinations

Ctrl Key

The Control key, located at the left-hand side of the PC*jr*'s keyboard, will place the keyboard in the control mode when it is pressed. In the control mode, certain keys can be pressed to perform designated operations. We have already seen how the Control-Alt-Del key combination will generate a system reset. The Control key can be used in combination with other PC*jr* keys to edit program lines. This will be discussed in Lesson 9.

Editing Keys

A number of keys on the PC*jr* keyboard are used to edit program lines. These are depicted in figure 5.5. The editing keys will be discussed in Lesson 9.

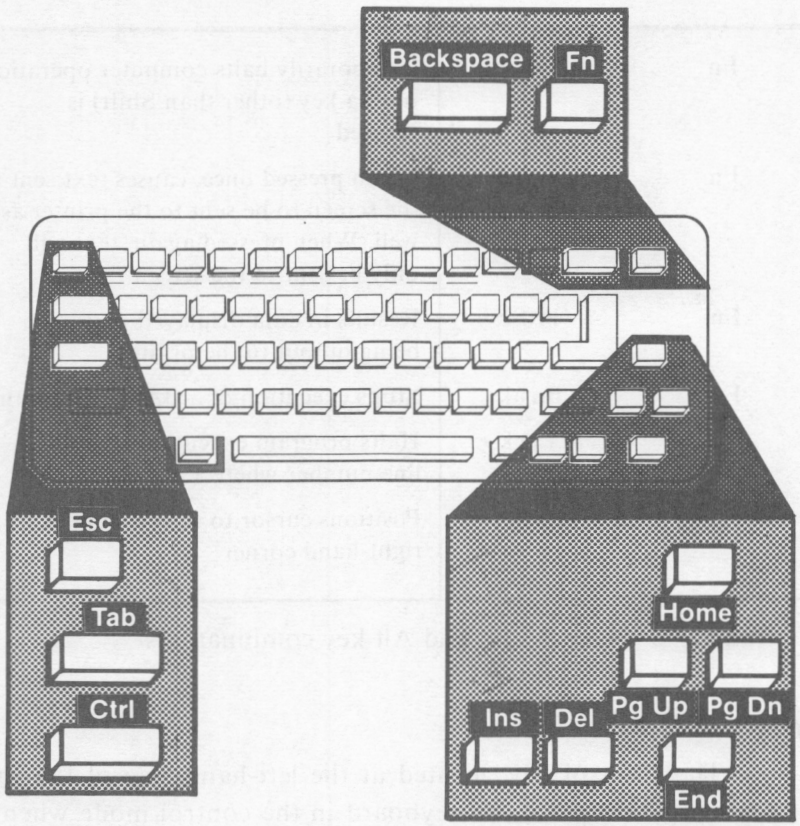


Figure 5.5. PC*jr* editing keys

Enter Key

The Enter key is used to send information from the PC*jr*'s keyboard to the computer. In most cases, the PC*jr* will not actually respond to a keyboard entry until the Enter key has been pressed.

For example, suppose that we made the following entries using the PC*jr*'s keyboard:

Nancy ↵

The display screen will appear as follows once the Enter key (↵)* has been pressed.

* In this book, the symbol ↵ will be used to indicate pressing the Enter key.

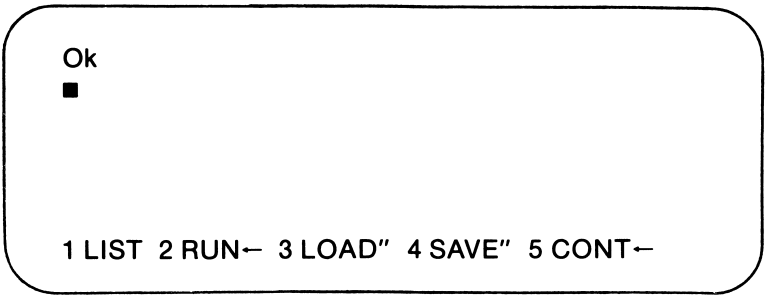
How to Use this Book -- Keyboard Entries

Throughout this book, you will notice words and numbers printed on the left-hand side of the page with a line underneath it. These characters are to be entered on the PC*jr*'s keyboard. The comments and illustrations to the right of the line describe what will happen once that entry has been made.

```
The IBM PC jr BASIC
Version J1.00
Copyright IBM Corp.  1981, 1982, 1983
59694 Bytes free
Ok
Nancy
Syntax error
Ok
_
```

BEEP ↵ The PC*jr*'s speaker sounded with a beep.

SCREEN 1 ↵ The display screen momentarily went blank, and then the following screen display appeared:

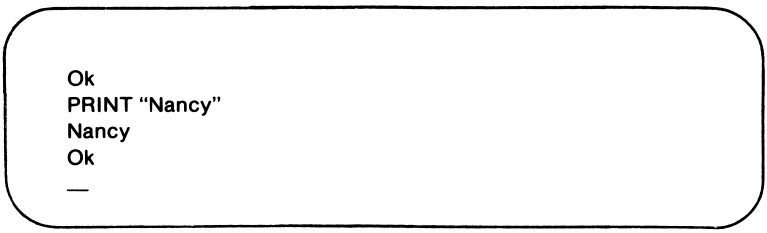


Notice that the cursor appears as a solid block rather than a flashing line.

COLOR 2 ↵ The screen's color changed to green.

SCREEN 0 ↵ The normal screen display reappears.

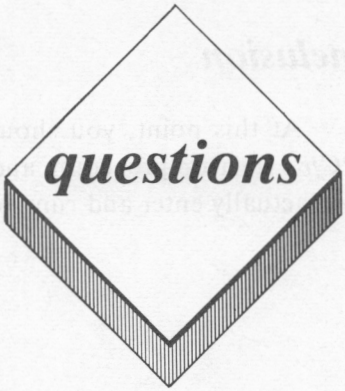
PRINT "Nancy" The display screen will appear as follows:



Notice that our entry did not generate an error message. Instead, the word "Nancy" was displayed on the screen.

Conclusion

At this point, you should have a good understanding of the PC*jr*'s start-up procedure and its keyboard. In the next lesson, you will actually enter and run your first BASIC program.



True or False

1. The Cartridge BASIC cartridge must be installed before the PC*jr* can be started up without using the disk drive.
2. The A> prompt indicates that a BASIC command can be entered.
3. A cold boot can be executed by pressing the Control, Alt, and Del keys simultaneously.
4. The alternate mode enables the operator to enter a BASIC command by pressing the Alt key simultaneously with one of the letter keys.
5. Pressing the Fn key causes the PC*jr* to be in the control mode.

Multiple Choice

1. The following system date entries are valid:
   A. 07.01.82
   B. 12-1-84
   C. 2:24:82
   D. All of the above
   E. None of the above

2. The following system time entry (entries) are valid:
   A. 7
   B. 17/34/48.12
   C. 8:11:17:42
   D. All of the above
   E. None of the above

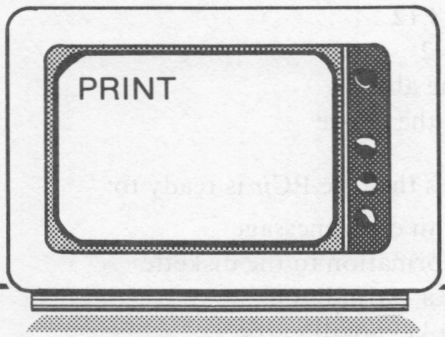
3. The flashing cursor indicates that the PC*jr* is ready to:
   A. Display an error message
   B. Send information to the diskette
   C. Perform a warm boot
   D. Accept a keyboard entry
   E. None of the above

4. The following key is used to send information from the PC*jr*'s keyboard to the computer:
   A. Fn key
   B. Ctrl key
   C. Alt key
   D. Shift key
   E. None of the above

5. The following component(s) is essential for practical PC*jr* usage:
   A. Keyboard
   B. Diskette drive
   C. Cassette recorder
   D. Cartridge BASIC
   E. All of the above

Computer Exercises

1. Start-up the PC*jr* without using the diskette drive.
2. Start-up the PC*jr* using the diskette drive.
3. Perform a warm boot.



Your First Program

lesson 6

Lesson Goals

- *Learn the difference between immediate and program mode entries*
- *Learn how to use BASIC's PRINT statement*
- *Learn how to execute a BASIC program*
- *Learn how to clear the display screen*
- *Learn how to list a program which is stored in memory*
- *Learn how to erase a program from memory*

Introduction

In this lesson, we will actually write and execute our first BASIC program. We will then learn how to use BASIC commands to clear the screen, list a program stored in memory to the screen, and erase a program from memory. Before learning these BASIC programming features, we will discuss the difference between the immediate and program modes in BASIC.

Immediate and Program Modes

In the **immediate mode,** a BASIC command entered using the PC*jr* keyboard will be executed when the Enter key is pressed. The following is an example of a BASIC command entry in the immediate mode:

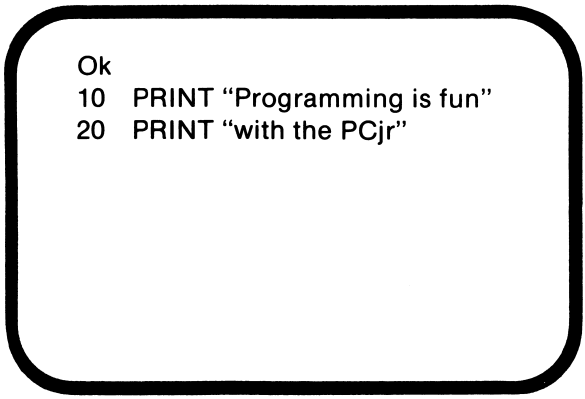
PRINT "Programming is fun" ↵ This entry will result in the following display:

```
Ok
PRINT "Programming is fun"
Programming is fun
Ok
_
```

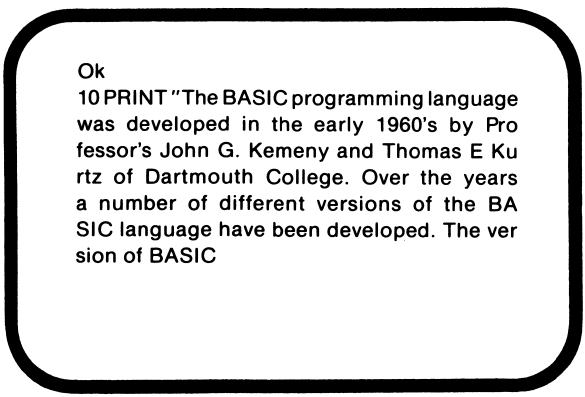
In the **program mode,** when program lines are entered, they are not executed but are instead stored for later execution. The stored program lines will be executed when BASIC's RUN command is entered in the immediate mode.

BASIC needs some way of identifying a program line entry as either being in the immediate mode or in the program mode. In the program mode, each separate line must be prefixed with a line number. When a BASIC command is entered with a line number preceding it, that line is known as a **program line.**

Program lines are ended and sent to the PC*jr*'s memory when the Enter key is pressed. Examples of program lines are given below:



The maximum number of characters that can be included in a program line is 255 including the character sent to memory when the Enter key is pressed. Since the PC*jr* can only include 40 characters on each display line, a program line can extend over 7 different display lines. This is shown in the following example:



BASIC executes program lines sequentially based upon their line numbers. In other words, if a program consisted of the following program lines,

```
 5 PRINT "New York 7"
 7 PRINT "St Louis 5"
 9 PRINT "Pittsburgh 7"
11 PRINT "Philadelphia 4"
13 PRINT "Pittsburgh is now in first place"
```

line 5 would be executed first, followed by lines 7, 9, 11, and 13, respectively.

Writing and Entering a Program

Now that we have learned some of the basic features of program lines and line numbers, we are ready to write our first BASIC program. Our first program will be a simple one. The purpose of the program will be to display the baseball standings in the National League's Eastern Division.

This program will use three BASIC commands, PRINT, REM, and END. The PRINT command is used to send information to the display screen. The information to be sent to the screen should be enclosed in quotation marks.

The REM statement is used to include the programmer's remarks in the program listing. Generally, these remarks are included to either describe the program's operation or its purpose.

The END statement is used to end program execution. Although END can be placed anywhere in a program, it is generally found on the final program line. Although a BASIC program will automatically stop execution when it finishes executing the final program line, it is still a good programming practice to include an END statement at the end of a BASIC program.

Now that we have some background on the BASIC commands that we'll be using, let's design our program to display the baseball standings. Let's assume that we want the standings displayed in the following format:

National League Eastern Division

Team	Wins	Losses
Pittsburgh	42	28
Montreal	41	29
Philadelphia	37	33
St. Louis	34	36
Chicago	29	41
New York	28	42

We already know that we can use the PRINT statement to display information on the screen. One of our obvious problems is to design our PRINT statements so that the information is displayed on the screen in a manner that is pleasant to read. This requires that we include the proper number of spaces in certain positions within each PRINT statement.

A grid such as that shown in figure 6.1 would help us accomplish this task. The first line to be displayed,

National League Eastern Division

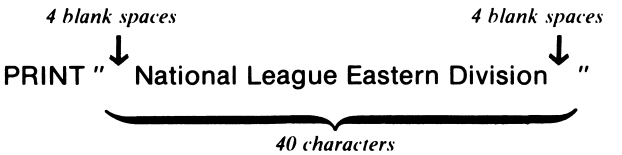
consists of 32 characters. We'll call it the title line. Since each display line allows 40 characters, by including 4 blank characters on one side of our text and 3 on the other in the PRINT statement, we could center it on the screen. This could be accomplished with the following program line:



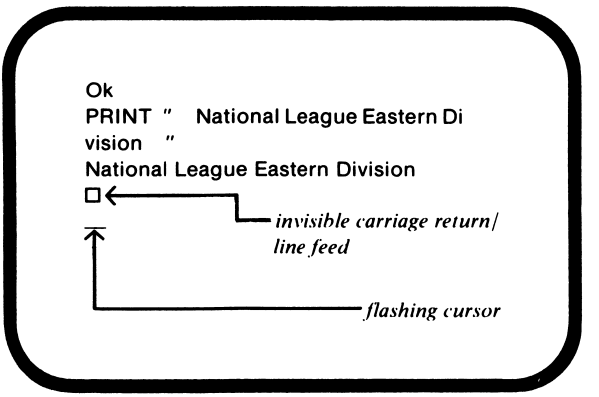
Notice that only 39 characters were enclosed within quotation marks even though the PC*jr*'s video display allows for a 40 character display. The reason for this is that the last character output by a PRINT statement is the carriage return/line feed. This character is invisible and can't actually be seen on the video display. However, it does occupy one position in the PRINT statement output.

The carriage return/line feed character tells the PC*jr* to begin displaying new data at the far left of the next screen line. Notice that when the PRINT statement is executed by itself, even though it does not display actual characters on the screen, it still outputs the carriage return/line feed character. This causes subsequent data to be output on the left-hand side of the next display line.

Therefore, if we issued a PRINT statement with an argument of 40 characters as follows,

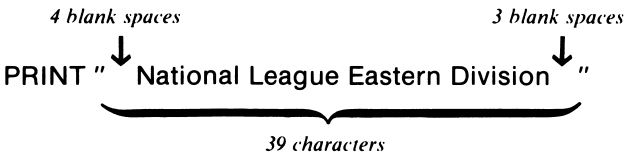


the carriage return/line feed character would actually be executed on the next display line.

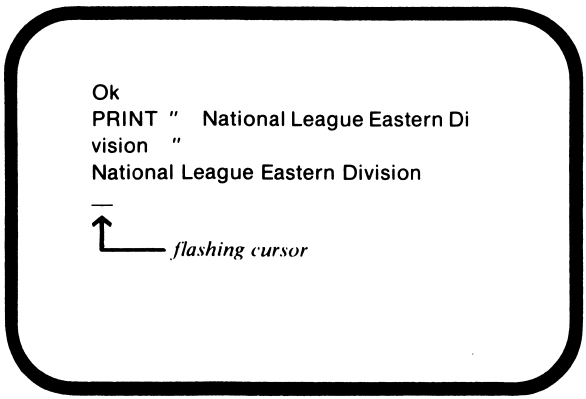


Notice that two lines were skipped on the display screen.

If we issued the PRINT statement with 39 characters rather than 40 characters, the carriage return/line feed will be output in the same line as the other characters, and only one line will be skipped. For instance, if the following immediate mode program line was input,



our output would appear as follows:



Suppose that we wanted a blank line to appear after the title line. As mentioned previously, if the PRINT statement is executed by itself, a blank line will be output. Therefore, our second line could be as follows:

```
20 PRINT
```

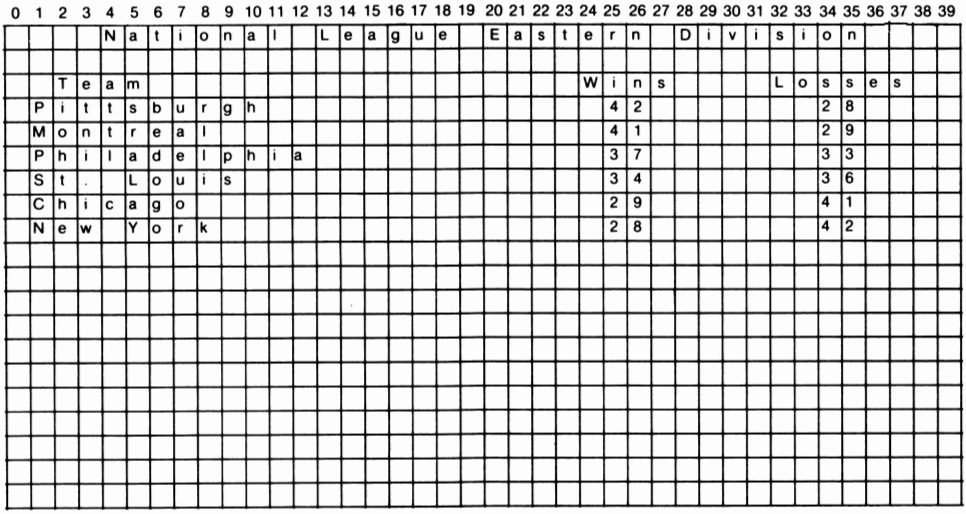


Figure 6.1 PRINT statement grid

Notice that we are numbering our lines in increments of 10. The use of consecutive line numbers (1,2,3,4,5,6,7,etc.) is generally not a good idea in a BASIC program. Suppose that after writing a program using consecutive line numbers,

```
21  PRINT "Pittsburgh"
22  PRINT "Montreal"
23  PRINT "St. Louis"
24  PRINT "Chicago"
25  PRINT "New York"
```

and we found that we needed to insert an additional line:

```
PRINT "Philadelphia"
```

If consecutive line numbers were used, a number of program lines would have to be renumbered to insert the new line.

```
21 PRINT "Pittsburgh"
22 PRINT "Montreal"
23 PRINT "Philadelphia"
24 PRINT "St. Louis"
25 PRINT "Chicago"
26 PRINT "New York"
```

These lines would have to be renumbered

By using non-consecutive line numbers, new program lines can be easily inserted.

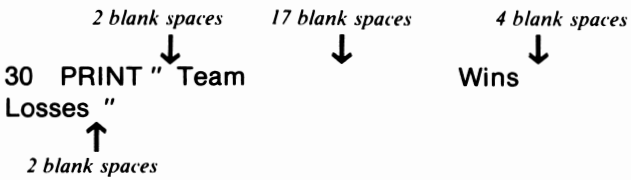
```
20 PRINT "Pittsburgh"
30 PRINT "Montreal"
35 PRINT "Philadelphia"
40 PRINT "St. Louis"
50 PRINT "Chicago"
60 PRINT "New York"
```

new program line inserted here →

Continuing with our program, our next task is to display the following line:

```
Team                    Wins    Losses
```

We'll call this line the headings line. Using the grid in figure 6.1, we could program this line as follows:



Our next step would be to write program lines to display each of the six teams with their respective number of wins and losses. Again, using the grid in figure 6.1, we could program these lines as follows:

```
                                 1 blank space      13 blank spaces      6 blank spaces
                                       ↓                  ↓                  ↓
40 PRINT " Pittsburgh                                                      42
4 blank spaces____28 ↑ "
50 PRINT " Montreal                                                        41
   29    "
60 PRINT " Philadelphia                                                    37
   33    "
70 PRINT " St. Louis                                                       34
   36    "
80 PRINT " Chicago                                                         29
   41    "
90 PRINT " New York                                                        28
   42    "
```

Now, we are ready to include the END statement to indicate that the program has finished.

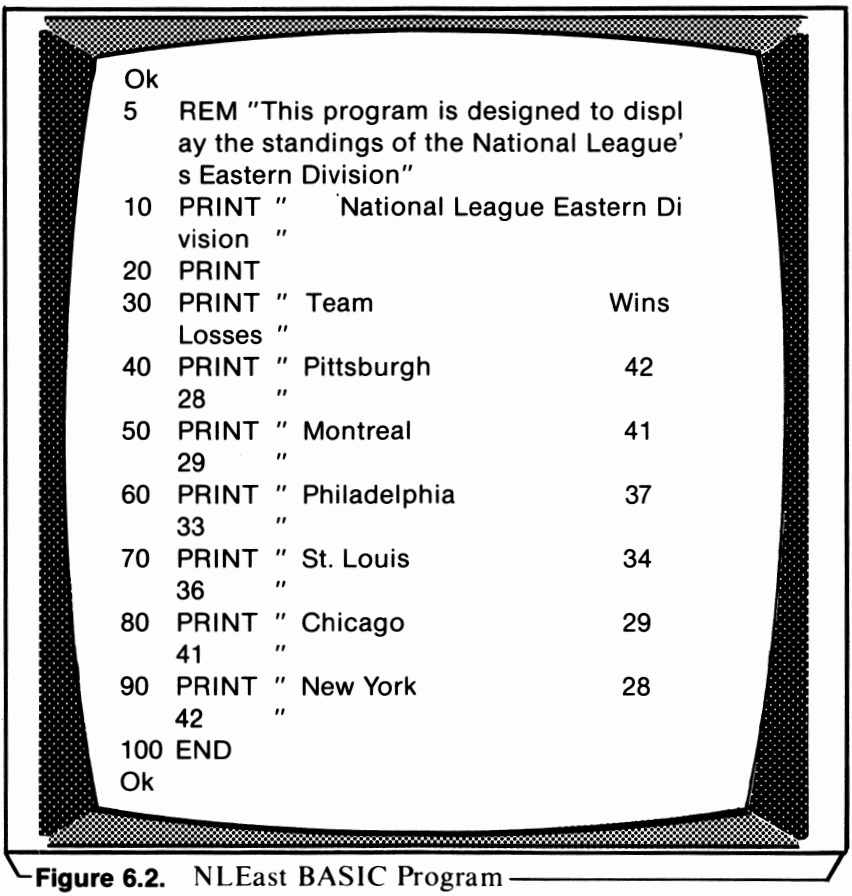
```
100 END
```

At this point, it might be helpful to include a REM statement in this program to describe its purpose. Let's include the following:

```
5 REM "This program is designed to displ
ay the standings of the National League'
s Eastern Division"
```

Our program would now appear as shown in figure 6.2.

Congratulations! If you have been following along, you have written and entered your first BASIC program. Our next step will be to run that program.



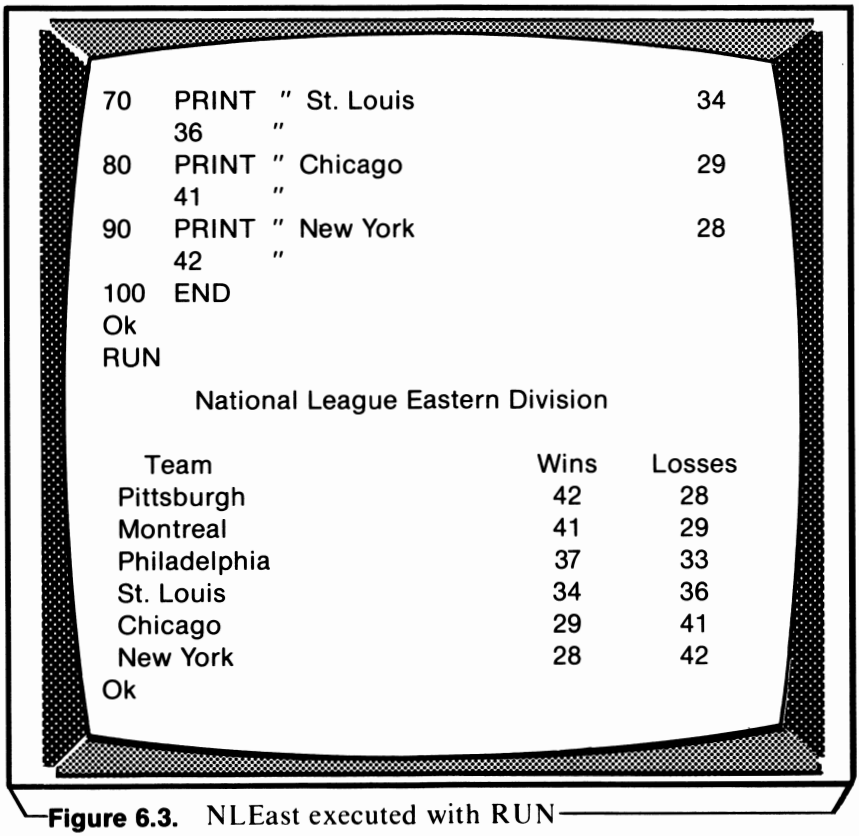
```
Ok
5   REM "This program is designed to displ
    ay the standings of the National League'
    s Eastern Division"
10  PRINT "     National League Eastern Di
    vision "
20  PRINT
30  PRINT " Team                    Wins
    Losses "
40  PRINT " Pittsburgh              42
    28     "
50  PRINT " Montreal                41
    29     "
60  PRINT " Philadelphia            37
    33     "
70  PRINT " St. Louis               34
    36     "
80  PRINT " Chicago                 29
    41     "
90  PRINT " New York                28
    42     "
100 END
Ok
```

Figure 6.2. NLEast BASIC Program

Running a BASIC Program

Running a BASIC program is fairly simple. BASIC's RUN command is used to begin execution of the program stored in the PC*jr*'s memory. When the RUN command is entered and the Enter key pressed, the program lines stored in the PC*jr*'s memory will be executed in order, beginning with the lowest line number.

Let's execute the RUN command for the program we just entered. We'll begin referring to this program as "NLEast". When we execute RUN, the PC*jr*'s screen should resemble that shown in figure 6.3.



```
70    PRINT  " St. Louis                    34
       36     "
80    PRINT  " Chicago                      29
       41     "
90    PRINT  " New York                     28
       42     "
100   END
Ok
RUN

          National League Eastern Division

     Team                        Wins     Losses
  Pittsburgh                     42       28
  Montreal                       41       29
  Philadelphia                   37       33
  St. Louis                      34       36
  Chicago                        29       41
  New York                       28       42
Ok
```

Figure 6.3. NLEast executed with RUN

Clearing the Screen

Now that your screen is filled with text, you might want to erase it. BASIC's CLS statement is used to clear the screen. Enter CLS via the PC*jr*'s keyboard and press the Enter key. Your screen is now blank except for the Ok prompt.

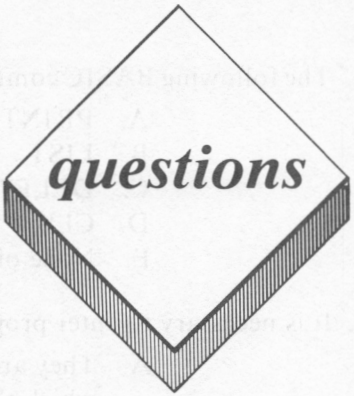
Listing the Program

Although we erased our NLEast program from the screen when we executed the CLS command in the immediate mode, we did not erase the program from the PC*jr*'s memory. BASIC's LIST command is used to display the program stored in memory on the screen. Enter LIST on the PC*jr*'s keyboard and press the Enter key. NLEast will now be displayed on the screen.

Erasing the Program

You've covered quite a bit so far in this lesson, and you are probably ready for a rest. Before we stop, let's cover one more BASIC command, NEW. The NEW command erases the BASIC program stored in the PC*jr*'s memory.

Let's test this by entering NEW in the immediate mode and pressing the Enter key. From the screen display, we have no clue whether NLEast has been erased from memory. By attempting to list NLEast to the screen, we can determine whether or not it had been erased from memory. Enter LIST and press the Enter key. Notice that NLEast was not listed to the screen. This is due to the fact that it had been erased when NEW was executed.



True or False

1. A program line cannot extend over more than one screen display line?
2. The END statement stops execution of a BASIC program.
3. When the CLS command is used, the program currently stored in the PC*jr*'s memory will be erased.
4. RUN is generally entered in the program mode.
5. If the END statement is not included in a BASIC program, that program will not stop executing.

Multiple Choice

1. The following statements regarding program lines are true:
   - A. Program lines are entered in the immediate mode.
   - B. Program lines need not contain a BASIC reserved word.
   - C. Program lines are executed sequentially.
   - D. None of the above
   - E. All of the above

2. The following BASIC command is used to clear the display screen:
   A. PRINT
   B. LIST
   C. DELETE
   D. CLS
   E. None of the above

3. It is necessary to enter program lines so that:
   A. They are entered in the order in which they are to be executed.
   B. They have been assigned consecutive line numbers.
   C. They will execute as they are entered.
   D. None of the above
   E. All of the above

4. Which of the following BASIC statements is generally used to describe a program's purpose or operation?
   A. PRINT
   B. LET
   C. REM
   D. CLS
   E. None of the above

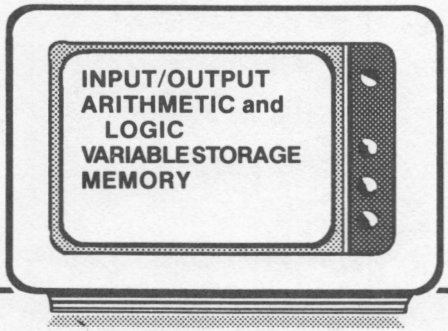
Computer Exercises

1. Write a program to display the following information:

Test Scores

Name	Percentage	Grade
Adams, William	89	B
Croghan, John	78	C+
Donner, Mary	94	A-
Gunderson, George	67	D
Matthews, Chris	98	A
Nagle, Reid	61	F
Vorhis, William	97	A

"Test Scores" should be centered in the middle of the display line. The first column title, "Name", should be centered within the space allotted for the display of the various names. The percentages and grades should be centered underneath their column headings.



How the PCjr Works

lesson 7

Lesson Goals

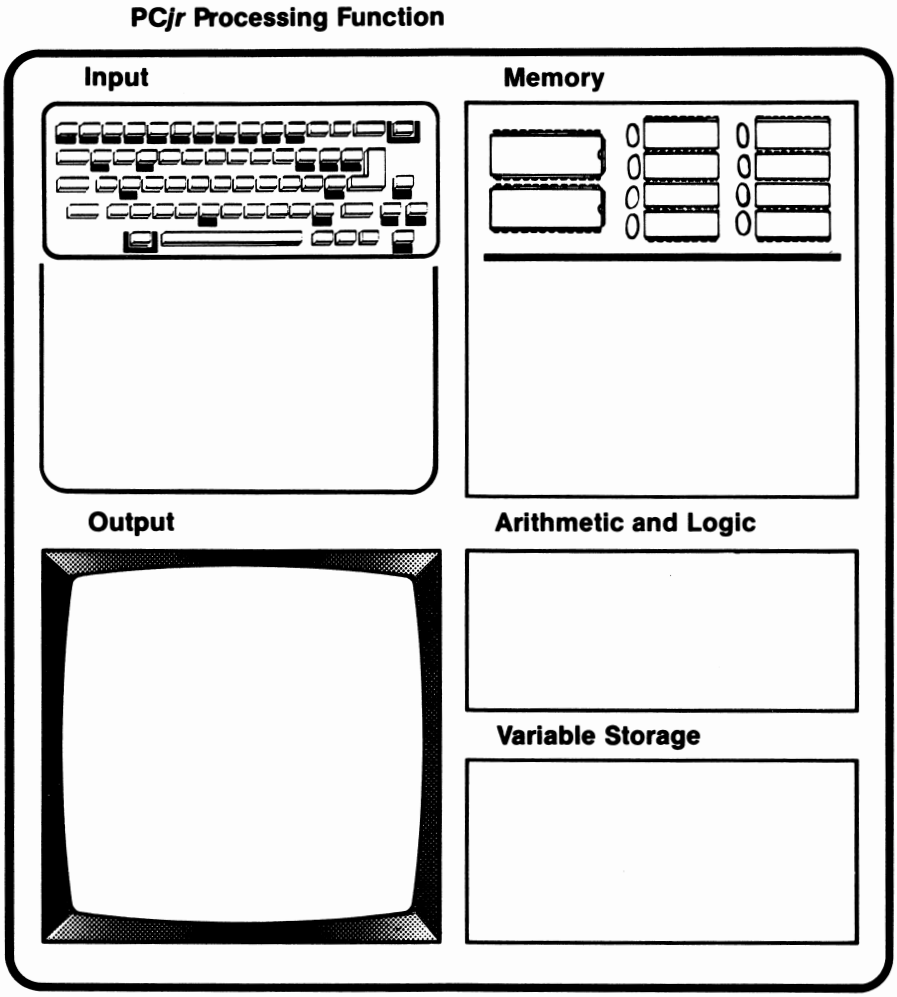
- *Understand the inner workings of the PCjr in its context as an information processing machine*
- *Learn what happens inside the PCjr when a BASIC program is entered, run, listed, and erased*

Introduction

In the last lesson, we learned how to enter and run a BASIC program. You may already have wondered what went on inside the PC*jr* while this program was being entered and run. In this lesson, we will explain how the PC*jr* executes a BASIC program as well as BASIC's LIST, CLS, and NEW commands.

We will not attempt to explain how the PC*jr* functions electronically. An understanding of the computer's electronic workings is not necessary to become computer literate. Instead, we will attempt to explain the workings of the PC*jr* in its function as an information processing machine. As we learned from our NLEast program, information processing consisted of entering information using the keyboard, processing the information inside the PC*jr*, and outputting the information so that it is visible to the user.

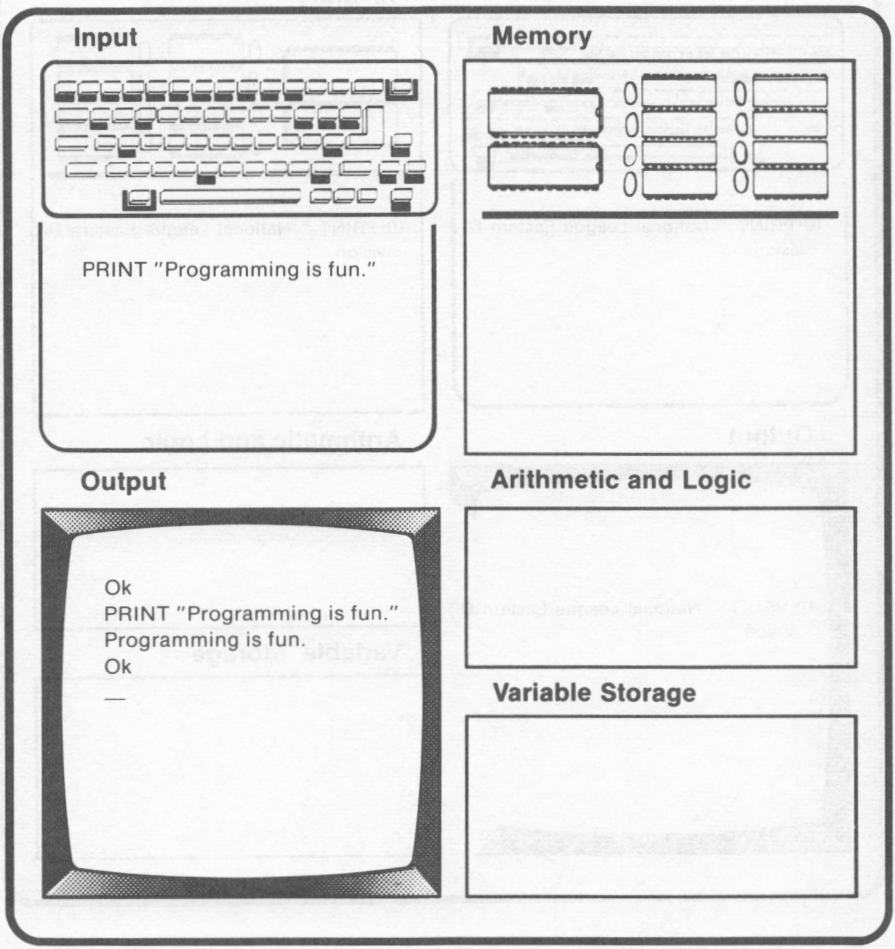
In this book, we will represent the input, output, and processing functions of the PC*jr* using the following illustration:



The input portion of this illustration is used to represent data input into the PC*jr*. Data is generally input using the PC*jr*'s keyboard. The memory area is used to indicate data which is present in the PC*jr*'s memory after the input. The arithmetic and logic area is used to indicate arithmetic and logical processing operations taking place inside the PC*jr*. We won't be concerned with this area until lesson 12. The variable storage area is used to store variable values. This will be discussed in lesson 11.

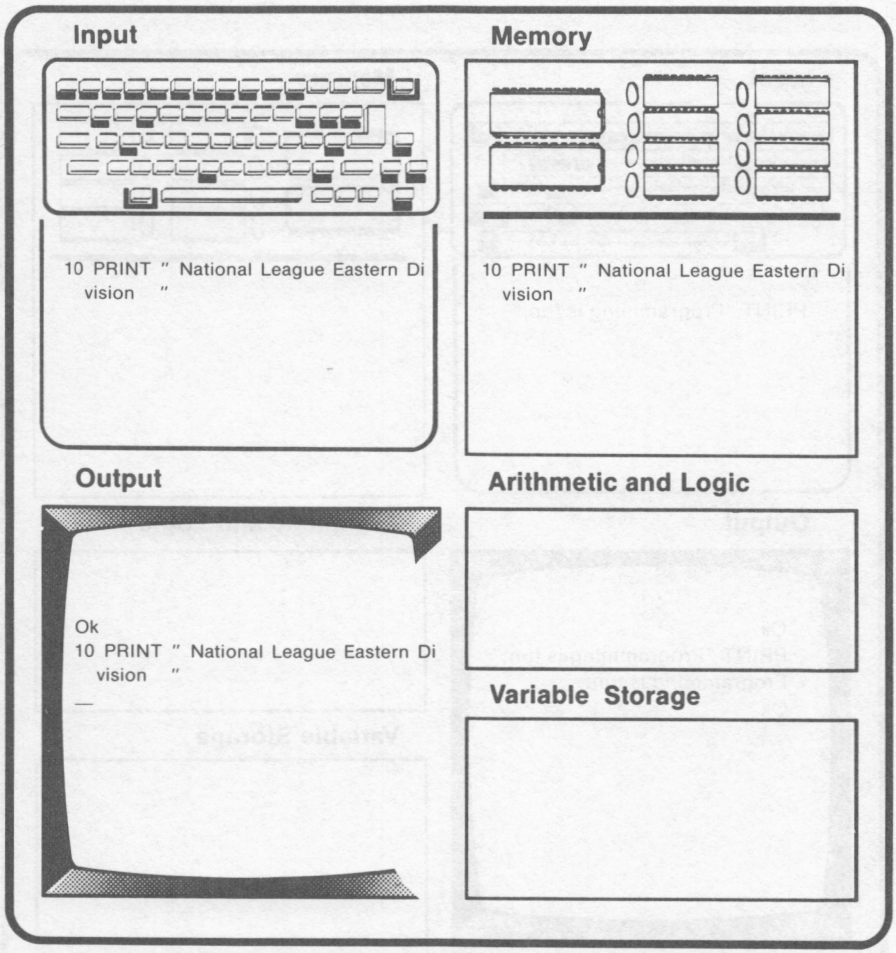
BASIC Program Entry

In this section, we will use our illustration representing the PC*jr*'s information processing procedure to show what happens when a BASIC program is entered. First, however, let's enter the following immediate mode BASIC command:

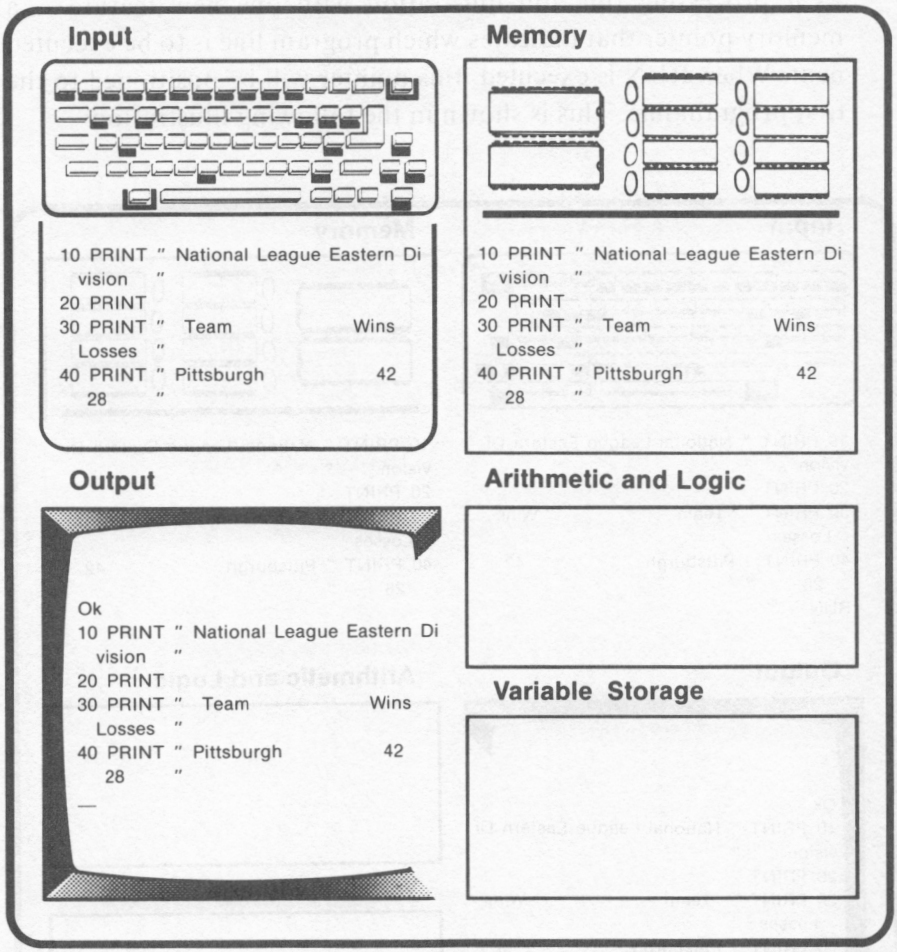


Notice that our immediate mode BASIC command did not utilize the PC*jr*'s memory portion. The PRINT command was executed, and the information it contained was displayed as output.

Now, let's enter the CLS command to clear the screen. Next, we'll enter the first line from NLEast (We'll ignore the REM statement).



Notice that the program line was both output to the screen and stored in memory. This process continues as we enter more program lines.

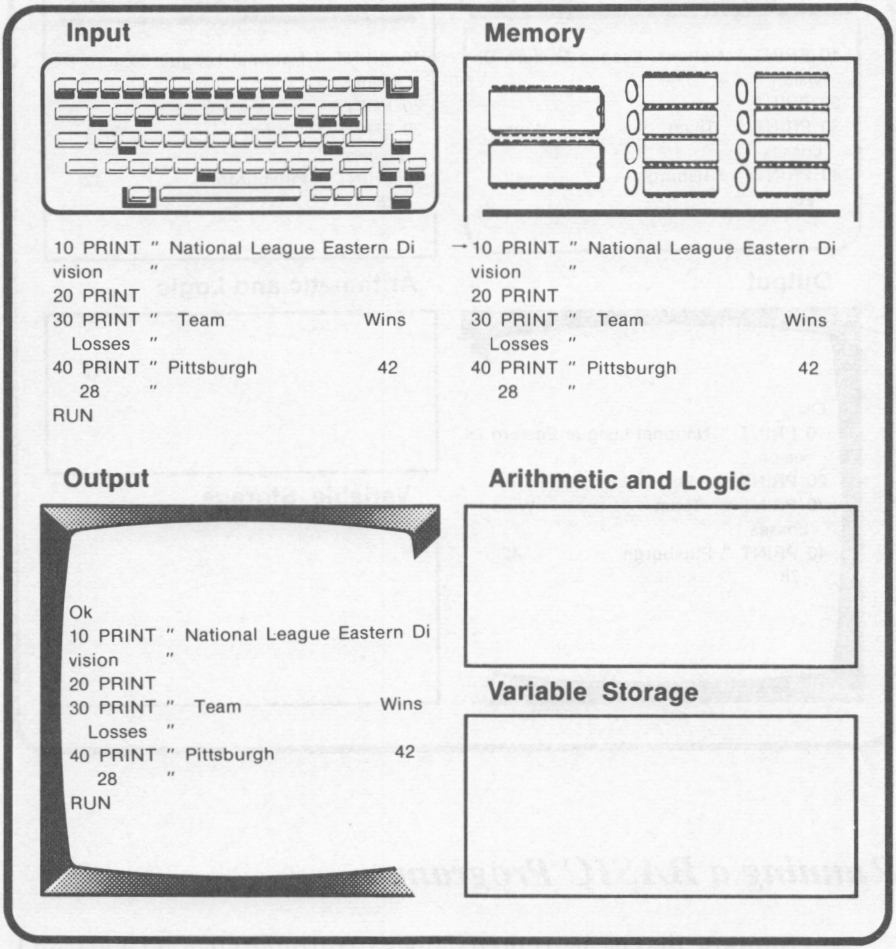


Running a BASIC Program

Suppose that we had entered lines 10 through 40 of NLEast, and we now wanted to run that portion of the program. We could do so by entering RUN.

Once RUN has been entered, outwardly the program will appear to be executed almost instantaneously. Actually, the program is

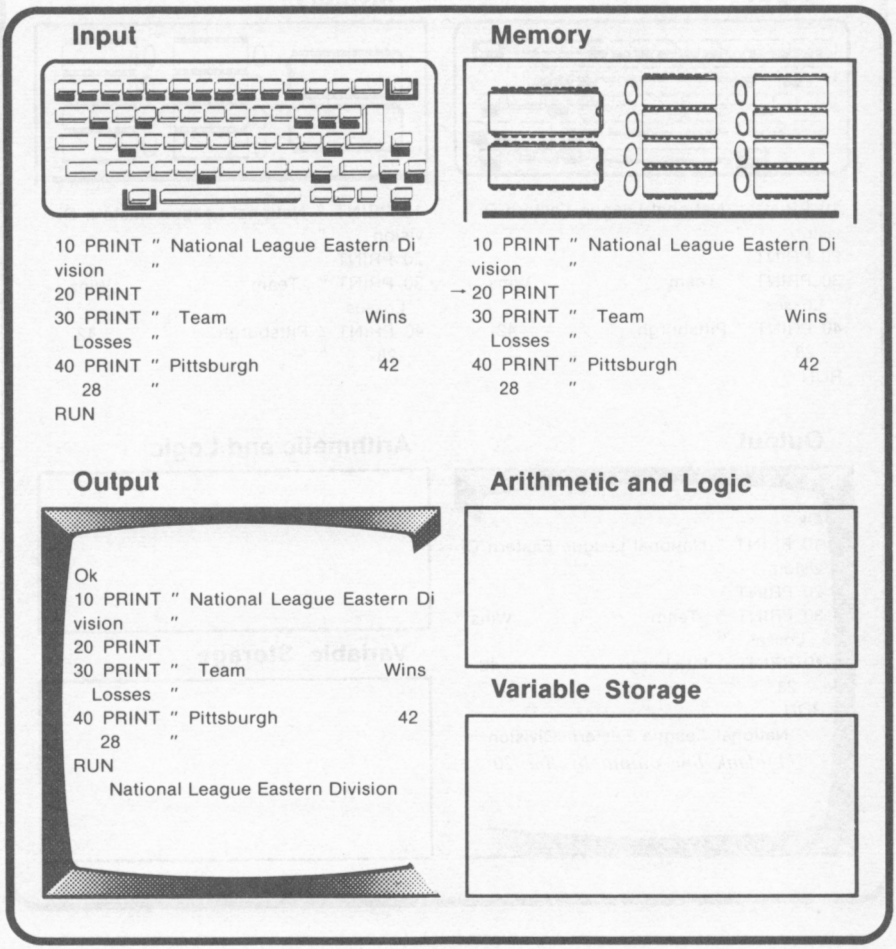
executed as a series of separate steps. We will describe these using our PC*jr* processing function illustration with one new feature -- a memory pointer that indicates which program line is to be executed next. When RUN is executed, this pointer will be positioned to the first program line. This is shown in the following illustration:



Notice that the cursor is not displayed in the Input section. This is due to the fact that the PC*jr* is not ready to accept a BASIC command entry. It is busy executing line 10 of the program in

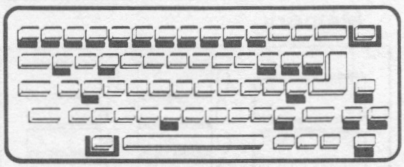
memory. Line 10 can be referred to as the **current line.** This is the line indicated by the memory pointer as the program line to be executed next.

When line 10 is executed, the PC*jr*'s output will change, and the line pointer will move to the next program line in memory. This is shown in the following illustration:



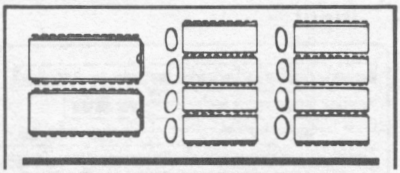
Program execution will continue in this same manner with lines 20, 30, and 40 as shown in the following illustrations. When line 20 is executed, a blank line will be output following "National League Eastern Division" in the PC*jr*'s output. The current line pointer will move to line 30.

Input



```
10 PRINT " National League Eastern Di
vision      "
20 PRINT
30 PRINT "  Team                Wins
  Losses    "
40 PRINT " Pittsburgh           42
   28       "
RUN
```

Memory



```
10 PRINT " National League Eastern Di
vision      "
20 PRINT
→30 PRINT "  Team                Wins
  Losses    "
40 PRINT " Pittsburgh           42
   28       "
```

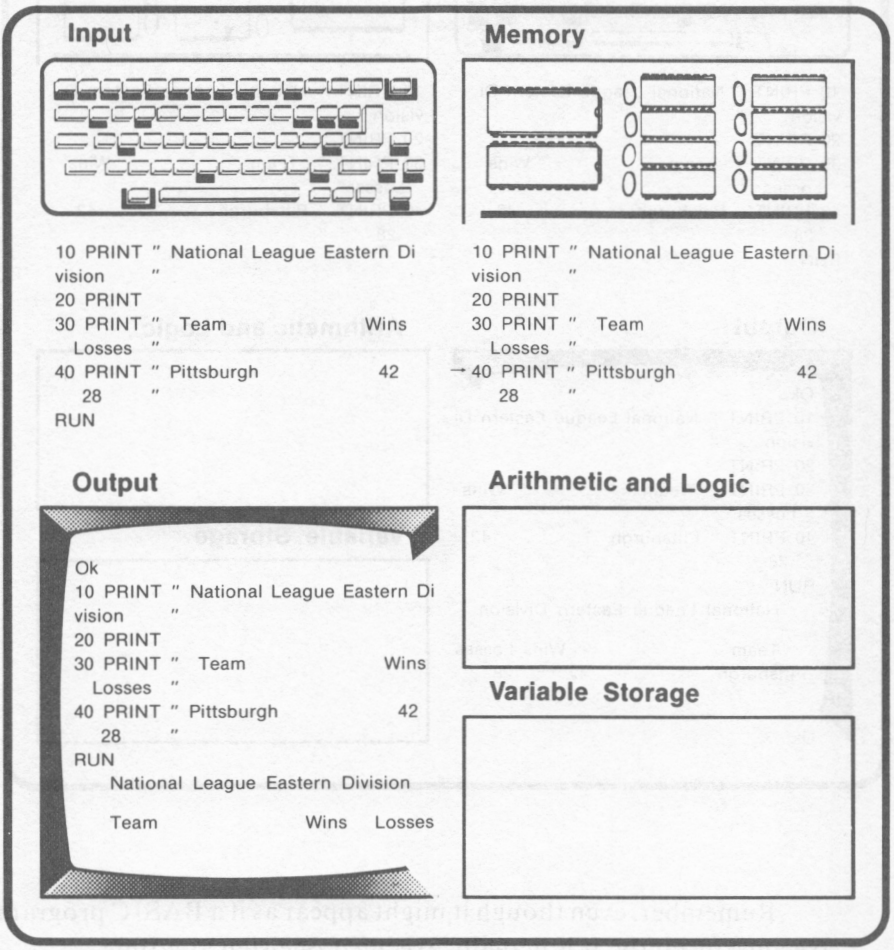
Output

```
Ok
10 PRINT " National League Eastern Di
vision      "
20 PRINT
30 PRINT "  Team                Wins
  Losses    "
40 PRINT " Pittsburgh           42
   28       "
RUN
    National League Eastern Division
   'l blank line output by line 20
```

Arithmetic and Logic

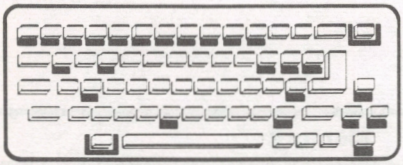
Variable Storage

When line 30 is executed, "Team Wins Losses" will be displayed in the PC*jr*'s output. The current line pointer will move to line 40.



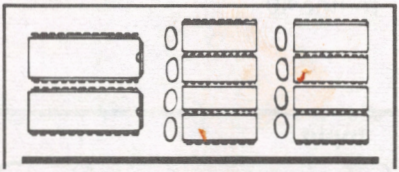
When line 40 is executed, "Pittsburgh 42 28" will be output. After line 40 has been executed, no more program lines remain in memory to be executed. The PC*jr* will output the BASIC prompt, Ok, on the screen along with the flashing cursor. This indicates that the PC*jr* is ready to accept a BASIC command entry.

Input



```
10 PRINT " National League Eastern Di
vision      "
20 PRINT
30 PRINT "  Team                  Wins
  Losses  "
40 PRINT " Pittsburgh             42
   28      "
RUN
```

Memory



```
10 PRINT " National League Eastern Di
vision      "
20 PRINT
30 PRINT "  Team                  Wins
  Losses  "
40 PRINT " Pittsburgh             42
   28      "
```

Output

```
Ok
10 PRINT " National League Eastern Di
vision      "
20 PRINT
30 PRINT "  Team                  Wins
  Losses  "
40 PRINT " Pittsburgh             42
   28      "
RUN
     National League Eastern Division

   Team                  Wins Losses
 Pittsburgh              42     28


Ok
_
```

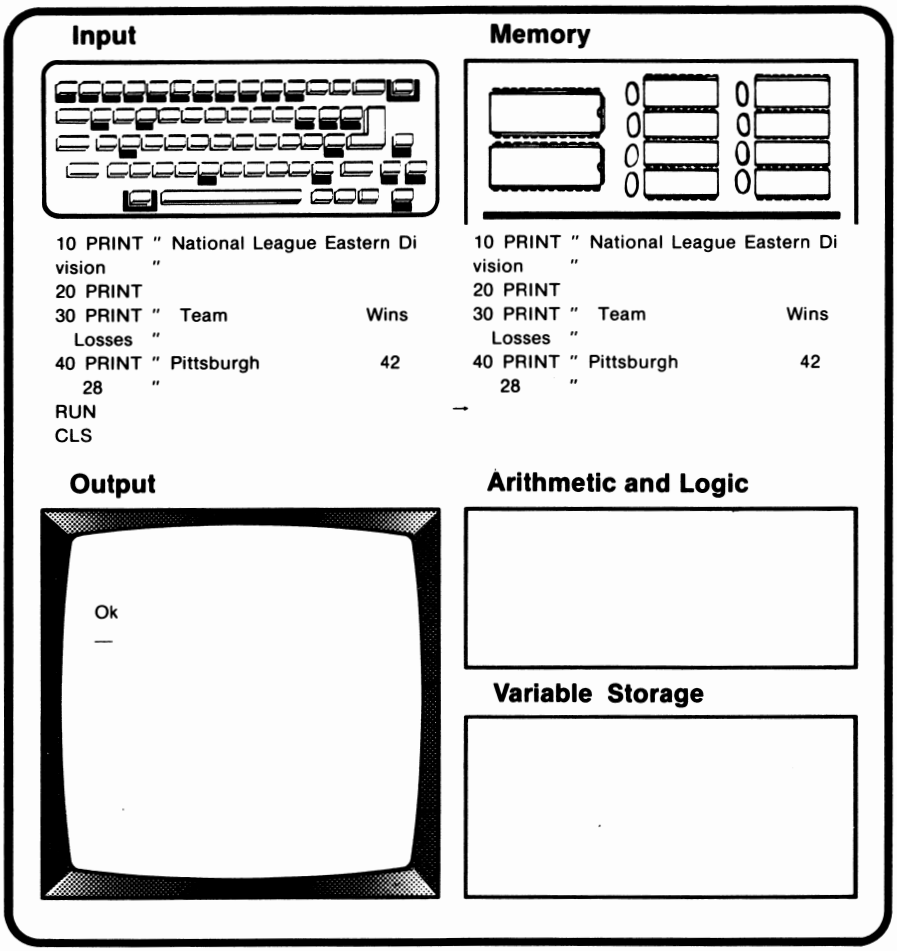
Arithmetic and Logic

Variable Storage

Remember, even though it might appear as if a BASIC program executes instantly, it is actually executed one step at a time.

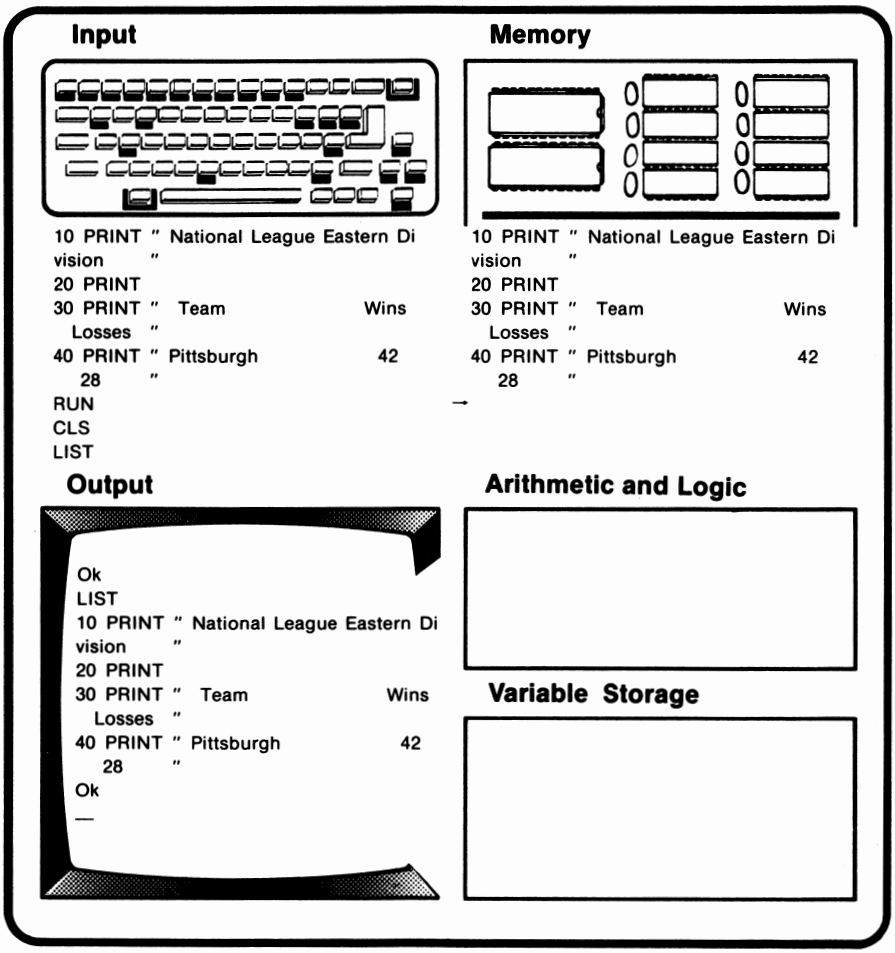
Clearing the Screen with a BASIC Program in Memory

Now that we have seen how the PC*jr* executes a BASIC program, let's examine what occurs when the CLS command is issued in the immediate mode to clear the screen. Suppose that we entered CLS. As shown in the following illustration, although the PC*jr*'s screen will be cleared, the BASIC program stored in its memory will remain intact.



Listing the BASIC Program

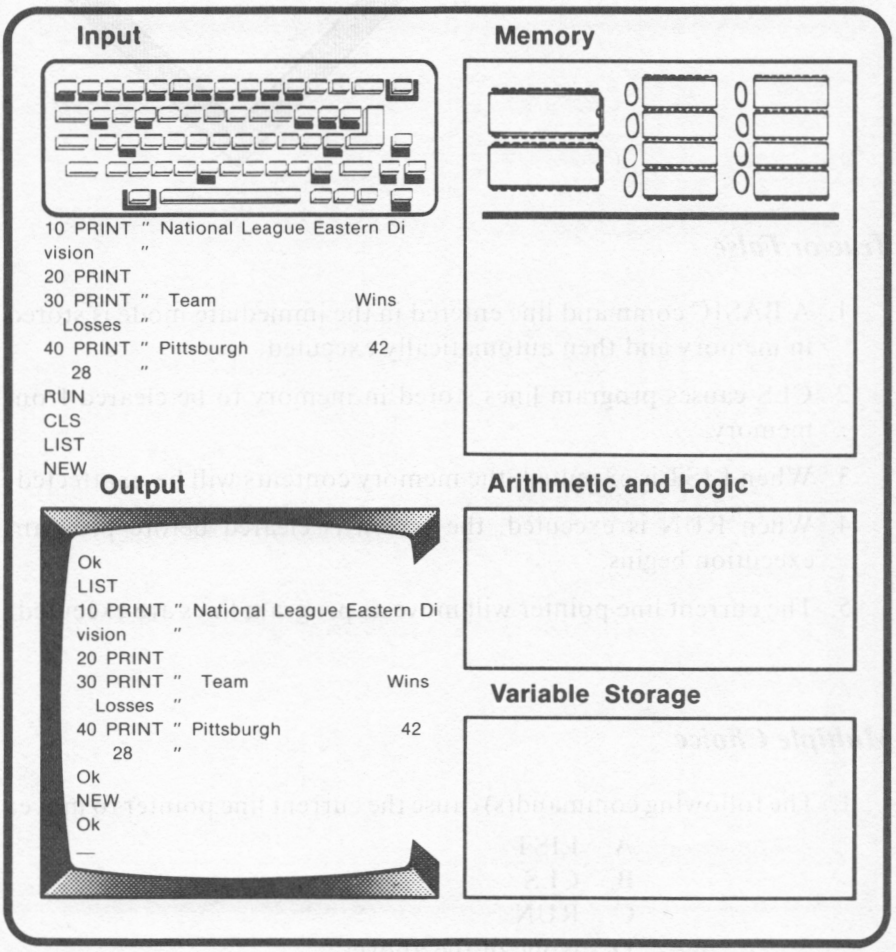
Let's examine what occurs when the PC*jr* executes the LIST command.

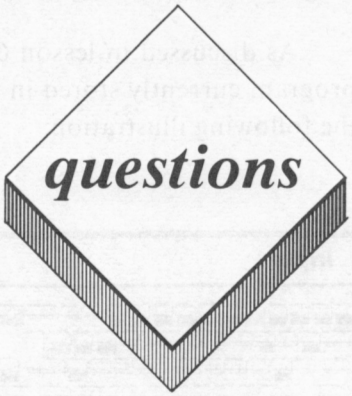


Notice that the program in memory is output, but that the memory contents are not affected.

Erasing the BASIC Program from Memory

As discussed in lesson 6, BASIC's NEW command erases the program currently stored in the PC*jr*'s memory. This is depicted in the following illustration:





True or False

1. A BASIC command line entered in the immediate mode is stored in memory and then automatically executed.
2. CLS causes program lines stored in memory to be cleared from memory.
3. When LIST is executed, the memory contents will be unaffected.
4. When RUN is executed, the screen is cleared before program execution begins.
5. The current line pointer will move as program lines are executed.

Multiple Choice

1. The following command(s) cause the current line pointer to move:
   A. LIST
   B. CLS
   C. RUN
   D. None of the above
   E. All of the above

2. The following command(s) can be used to determine whether or not a program is stored in memory:

   A. CLS
   B. PRINT
   C. LIST
   D. NEW
   E. None of the above

3. Any of the following affect the contents of the PC*jr*'s memory:

   A. Program mode entry
   B. RUN
   C. CLS
   D. None of the above
   E. All of the above

4. The following command(s) can affect the information displayed on the PC*jr*'s screen:

   A. CLS
   B. RUN
   C. LIST
   D. None of the above
   E. All of the above

5. When a BASIC program is executed, the PC*jr*:

   A. Executes all program lines simultaneously and sends the results to the screen.
   B. Executes each program line separately and then erases that line in memory after it has been executed.
   C. Temporarily erases existing information on the screen
   D. None of the above
   E. All of the above

Essay

1. Describe the current line pointer's function.
2. Describe the procedure you would use to determine whether or not

a BASIC program was stored in the PC*jr*'s memory.

3. Describe the procedure followed by the PC*jr* when the program you wrote for the computer exercise on page 128 is executed.



RUN, LIST, AUTO, RENUM & DELETE

lesson 8

Lesson Goals

- *Learn how to execute RUN with its optional line number parameter.*
- *Learn how to execute LIST with its optional line number parameters*
- *Learn how to generate new line numbers automatically using BASIC's AUTO command*
- *Learn how to renumber a program's line numbers using BASIC's RENUM command*
- *Learn how to delete one or more program lines using BASIC's DELETE command*

Introduction

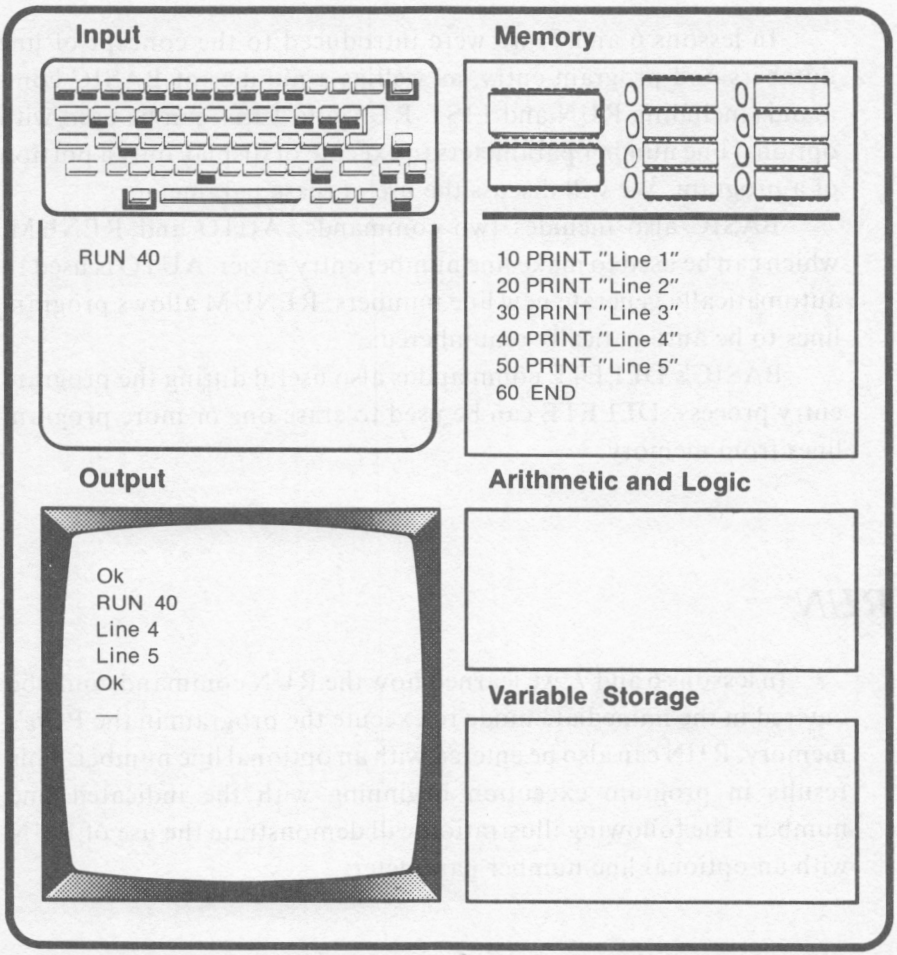
In lessons 6 and 7, we were introduced to the concept of line numbers and program entry, as well as a number of BASIC commands including RUN and LIST. RUN and LIST can be used with optional line number parameters to execute or display only a portion of a program. We will discuss the use of these parameters.

BASIC also includes two commands, AUTO and RENUM, which can be used to make line number entry easier. AUTO is used to automatically generate new line numbers. RENUM allows program lines to be automatically renumbered.

BASIC's DELETE command is also useful during the program entry process. DELETE can be used to erase one or more program lines from memory.

RUN

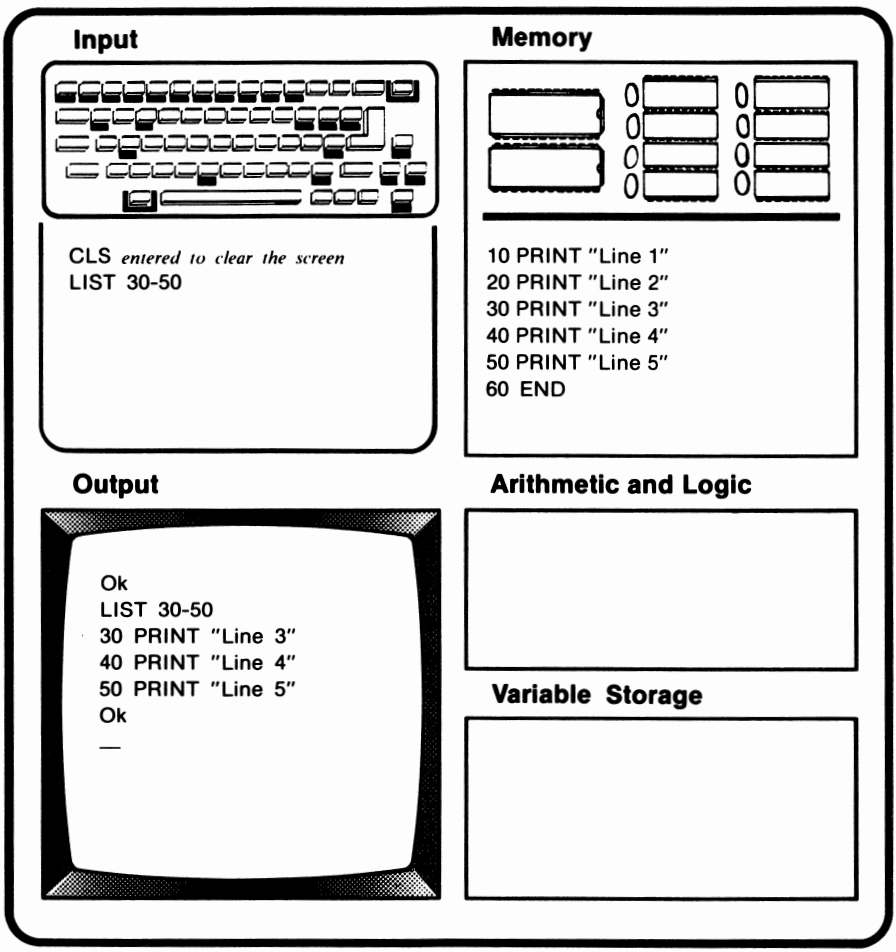
In lessons 6 and 7, we learned how the RUN command could be entered in the immediate mode to execute the program in the PC*jr*'s memory. RUN can also be entered with an optional line number. This results in program execution beginning with the indicated line number. The following illustration will demonstrate the use of RUN with an optional line number parameter:



Notice that executing the RUN command with the optional line number, 40, causes the BASIC interpreter to ignore lines 10, 20, and 30. Lines 40 and 50 were executed as evidenced by the output.

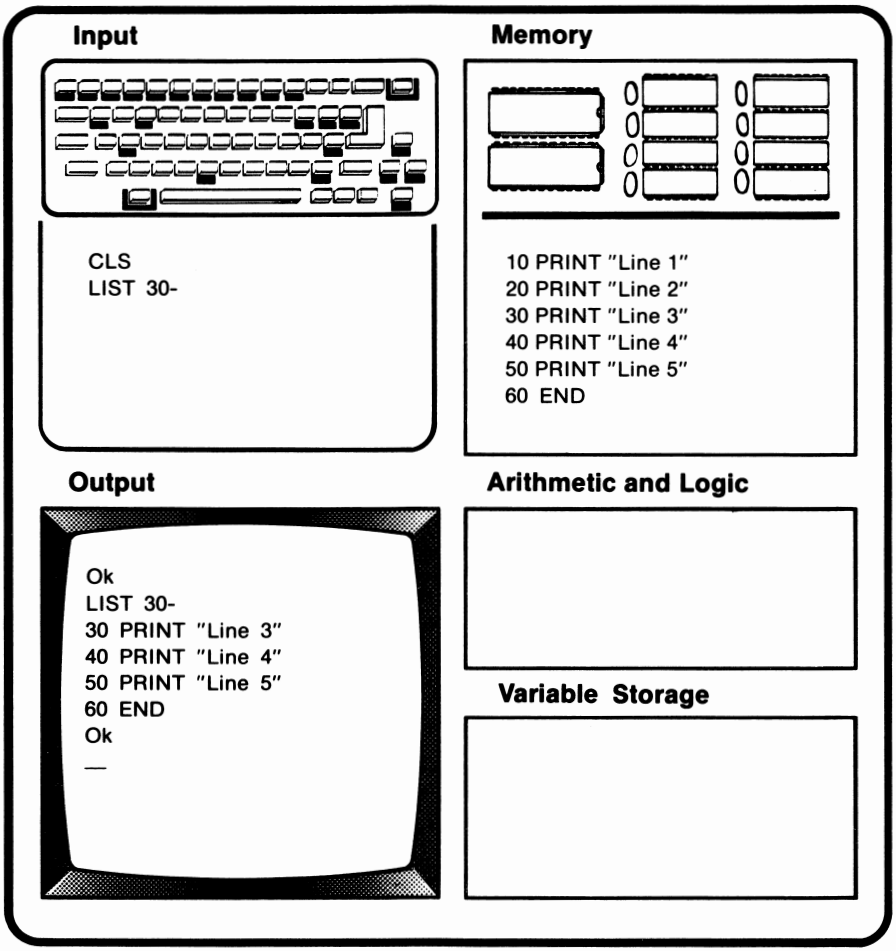
LIST

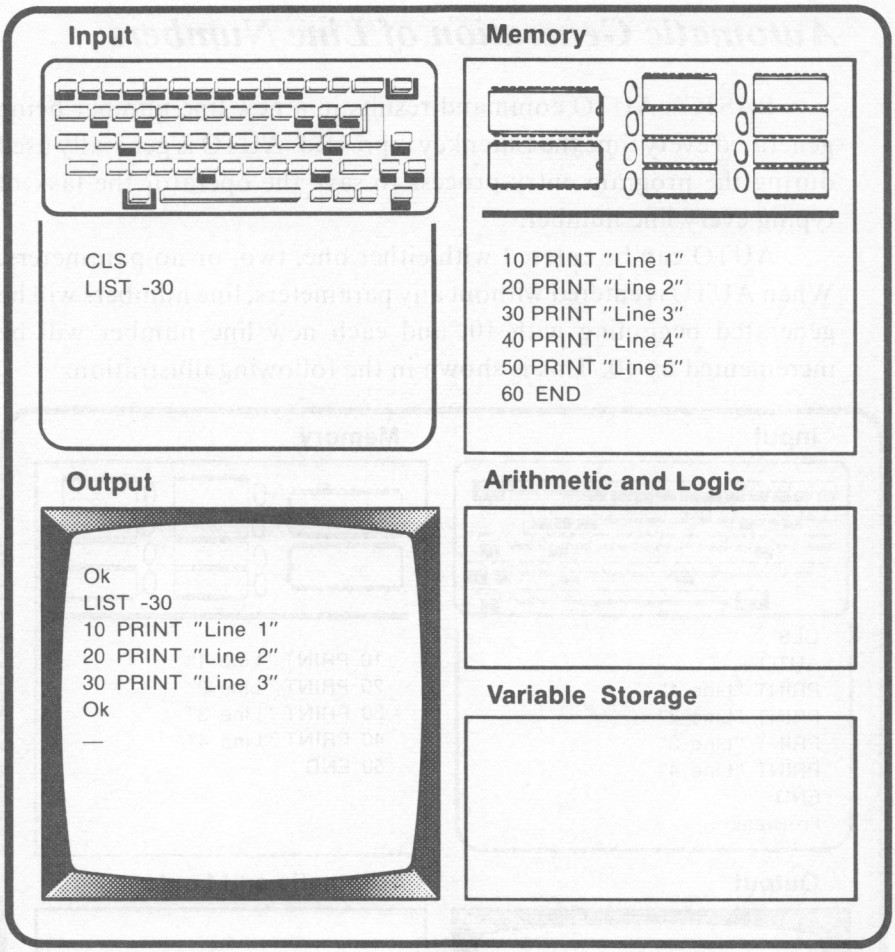
As we learned in lessons 6 and 7, the LIST command is used to display the BASIC program stored in the PC*jr*'s memory. LIST can be executed with optional line number parameters to display only a portion of the program stored in memory. This is shown in the following illustration:



As evidenced in the preceding illustration, when two line numbers are entered as LIST command parameters, all lines with values within the range indicated by these parameters will be displayed. Notice that the two line number parameters are separated with a dash (-).

LIST can also be executed with one line number parameter. Examples are shown in the following illustrations:





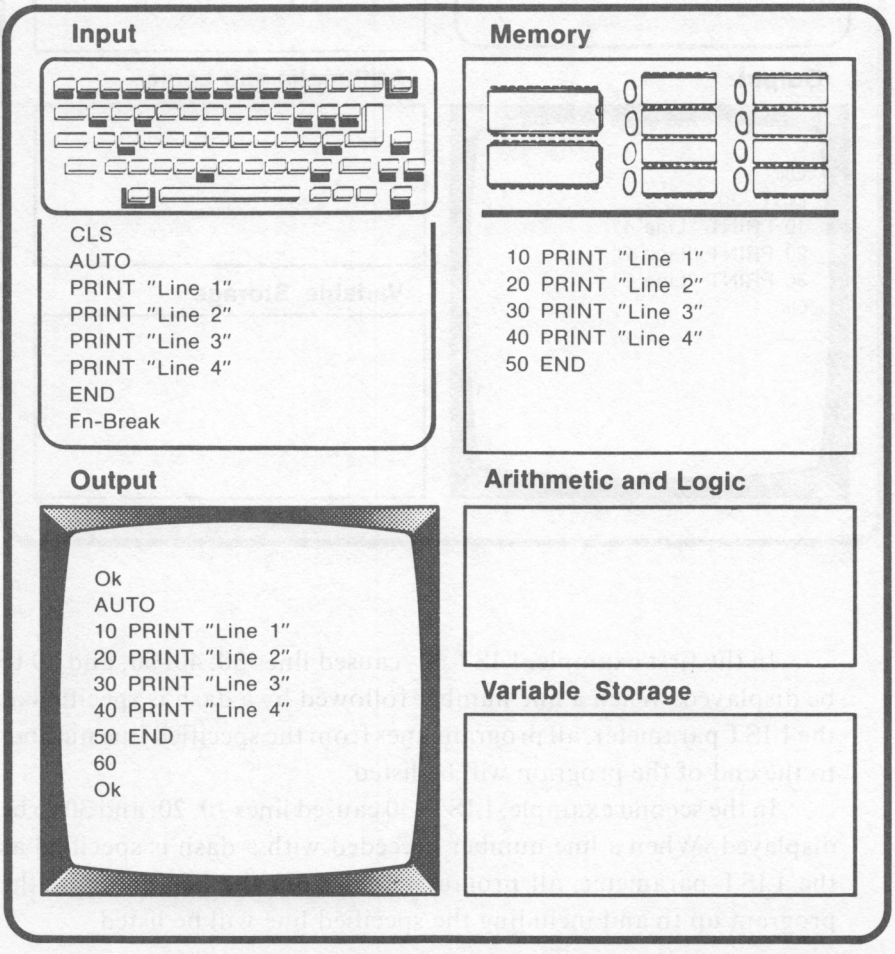
In the first example, LIST 30- caused lines 30, 40, 50, and 60 to be displayed. When a line number followed by a dash is specified as the LIST parameter, all program lines from the specified line number to the end of the program will be listed.

In the second example, LIST -30 caused lines 10, 20, and 30 to be displayed. When a line number preceded with a dash is specified as the LIST parameter, all program lines from the beginning of the program up to and including the specified line will be listed.

AUTO --
Automatic Generation of Line Numbers

BASIC's AUTO command results in a new line number being generated every time the Enter key is pressed. AUTO is generally used during the program entry process to save the operator the task of typing every line number.

AUTO can be entered with either one, two, or no parameters. When AUTO is entered without any parameters, line numbers will be generated beginning with 10, and each new line number will be incremented by 10. This is shown in the following illustration:

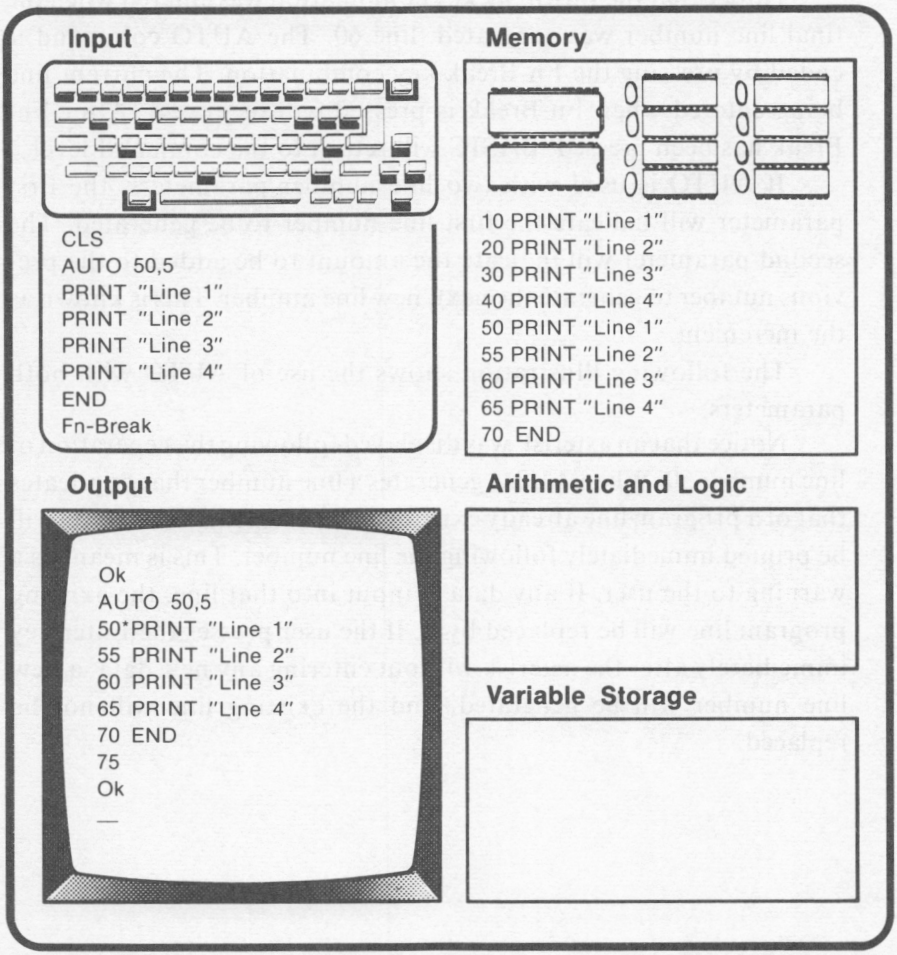


Notice that the Fn-Break key combination was entered when the final line number was generated, line 60. The AUTO command is ended by pressing the Fn-Break key combination. The current line being entered when Fn-Break is pressed will be erased. After Fn-Break has been pressed, BASIC will return to the command level.

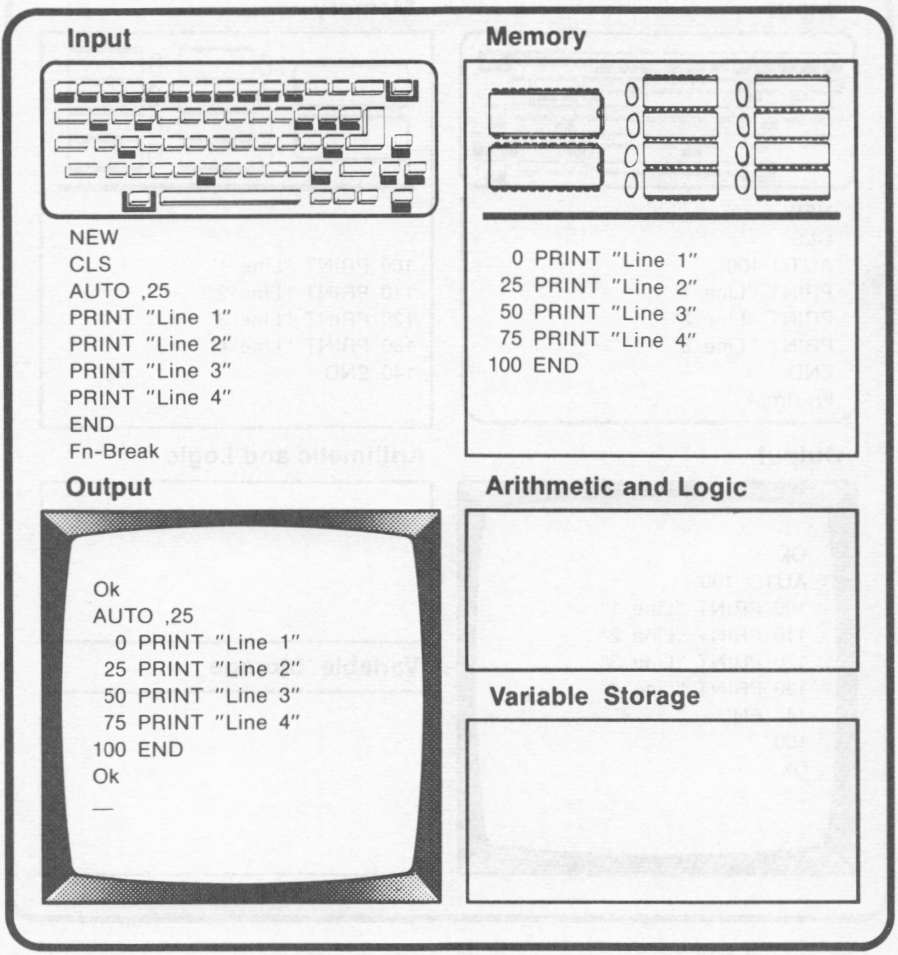
If AUTO is used with two line number parameters, the first parameter will indicate the first line number to be generated. The second parameter will indicate the amount to be added to the previous number to generate the next, new line number. This is known as the increment.

The following illustration shows the use of AUTO with both parameters.

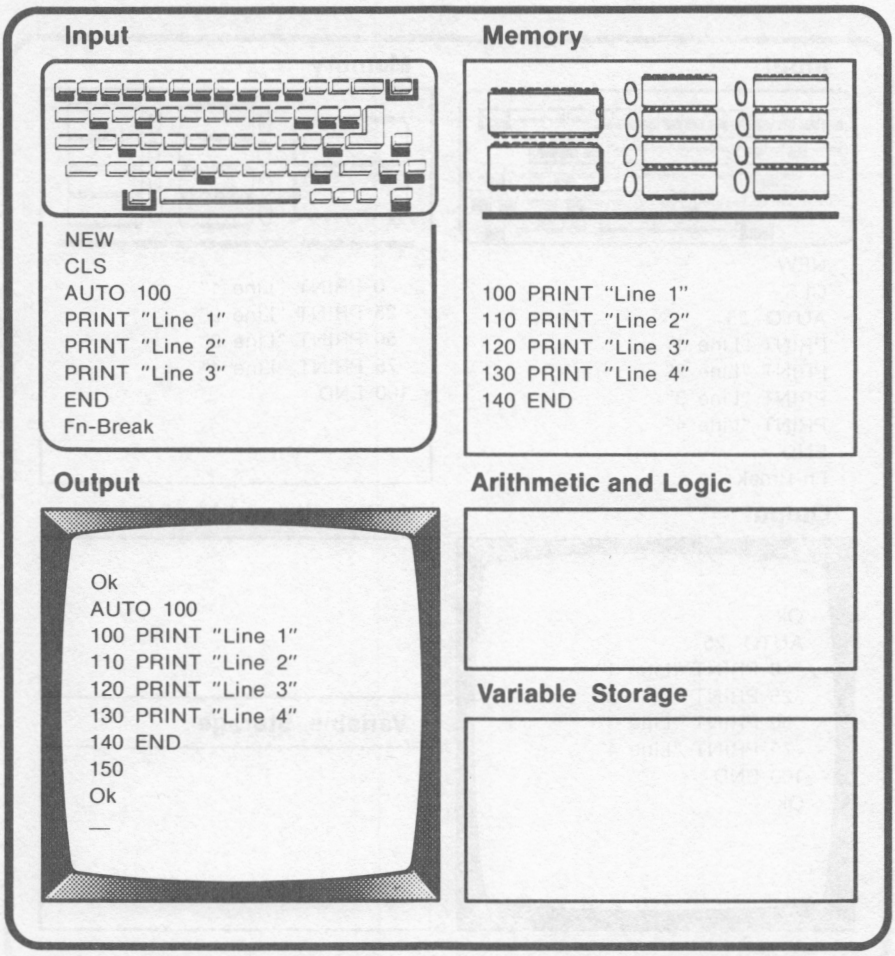
Notice that an asterisk was displayed following the generation of line number 50. When AUTO generates a line number that duplicates that of a program line already existing in memory, an asterisk (*) will be printed immediately following the line number. This is meant as a warning to the user. If any data is input into that line, the existing program line will be replaced by it. If the user presses the Enter key immediately after the asterisk without entering any new data, a new line number will be generated, and the existing line will not be replaced.



If only one parameter is indicated with AUTO and that parameter is preceded by a comma, the starting line number will be line 0. Each new line number will be incremented by the amount specified by the parameter. This is shown in the following illustration:

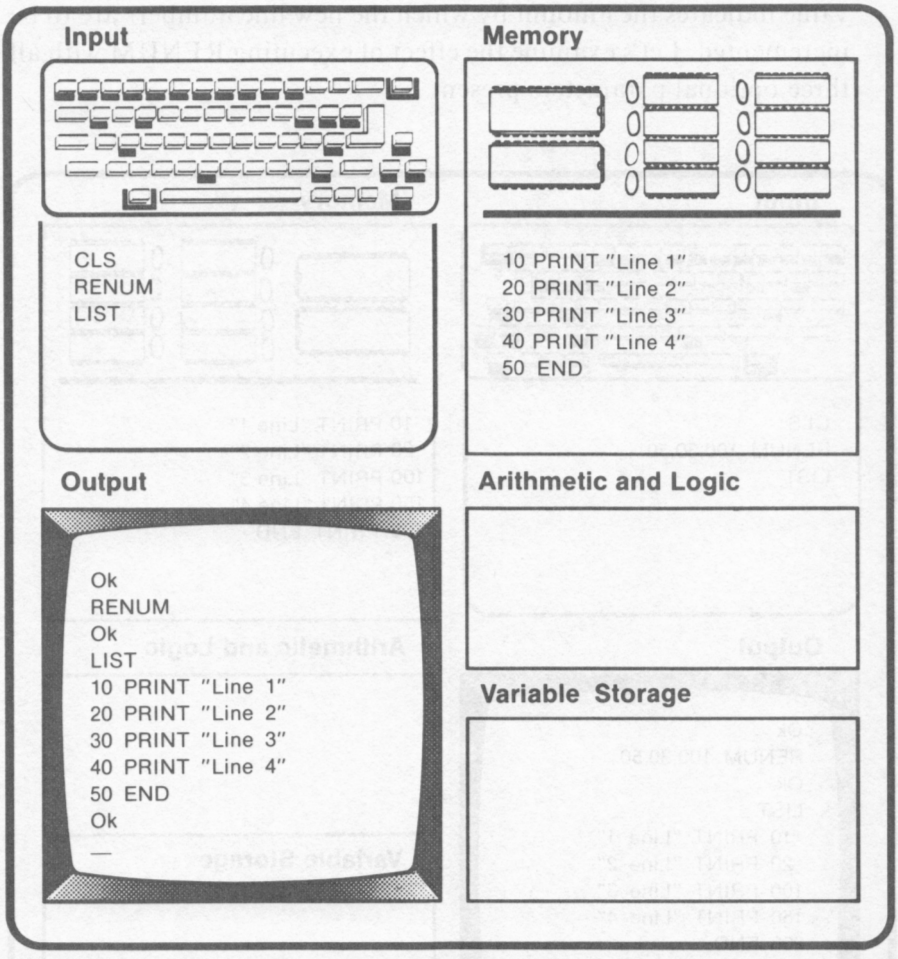


When a single line number parameter without a comma is indicated with AUTO, that number will be the first line number generated. The increment used for new line numbers will be 10.



RENUM -- Renumbering Program Lines

BASIC's RENUM command is used to renumber a program's line numbers. RENUM can be used with one, two, three, or no parameters. When RENUM is executed without any of its optional parameters, the program in memory will be renumbered with its first line numbered as 10. Successive lines will be renumbered in increments of 10. This is shown in the following example:

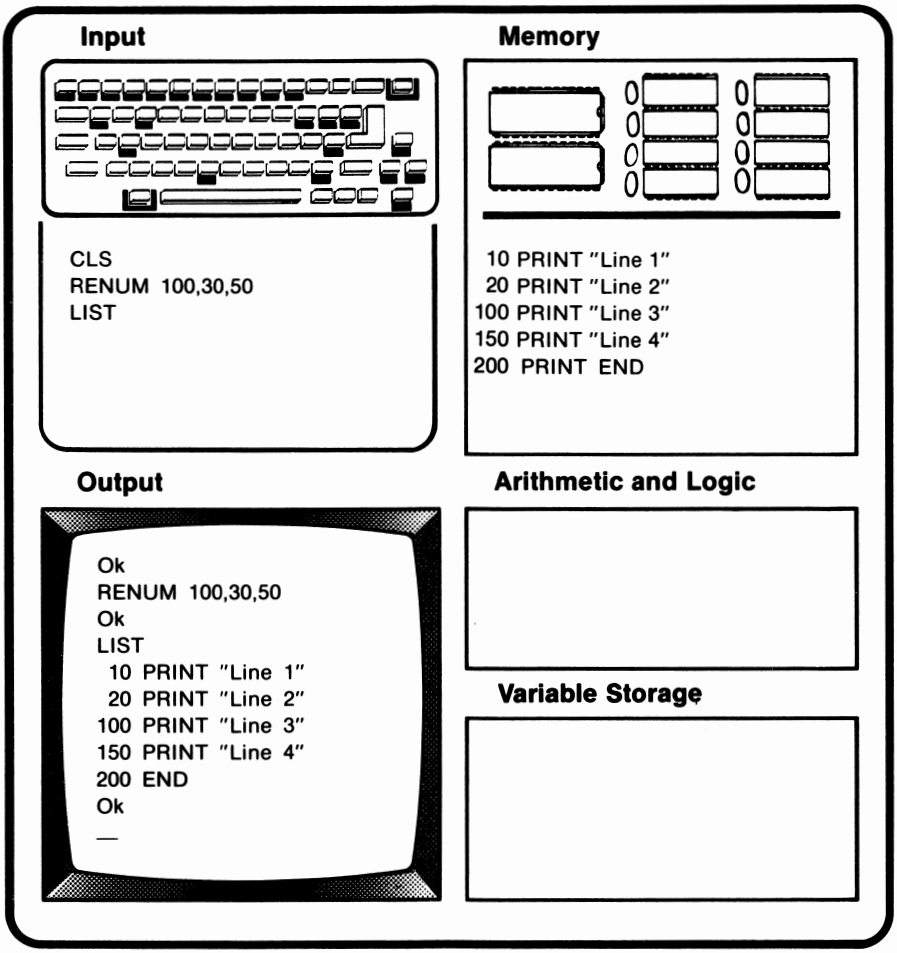


If RENUM is executed with all three parameters as shown below,

RENUM 100,30,50

the first parameter indicates the first new line number to be used in the renumbering process. The second value indicates the first line in the program where the renumbering process is to begin. The third

value indicates the amount by which the new line numbers are to be incremented. Let's examine the effect of executing RENUM with all three optional parameters present.

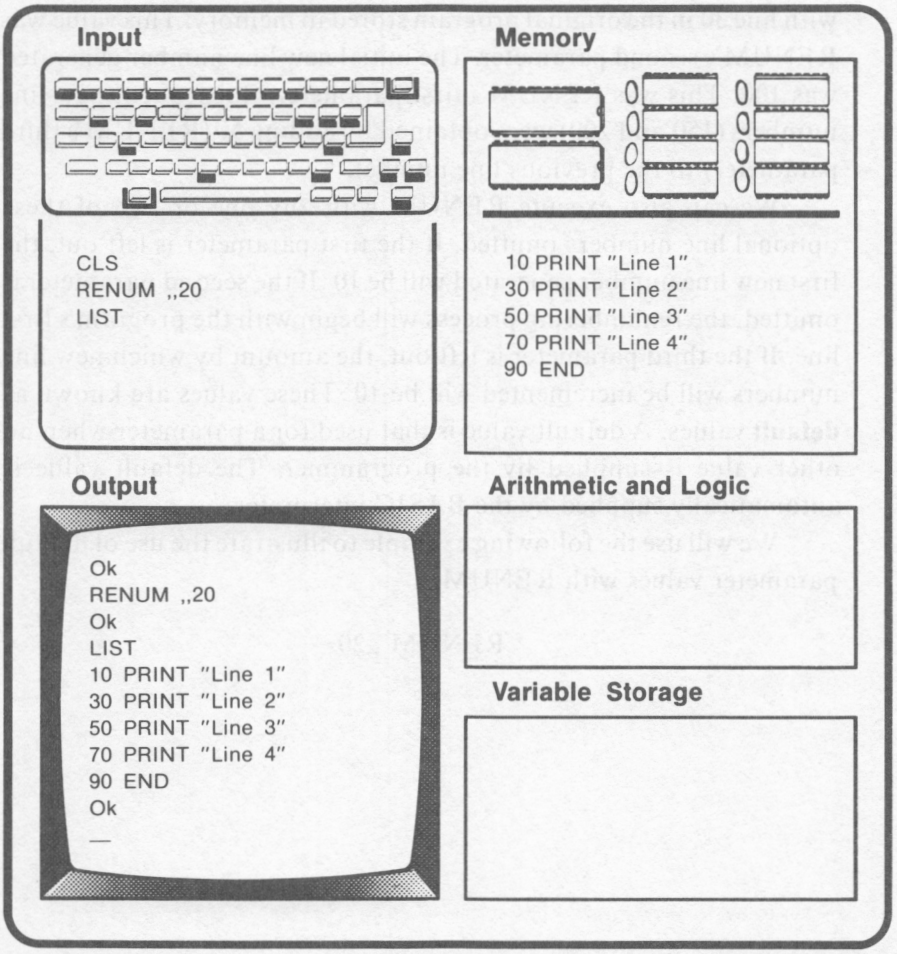


Notice from the example that program line renumbering began with line 30 in the original program stored in memory. This value was RENUM's second parameter. The initial new line number generated was 100. This was RENUM's first parameter. Subsequent new line numbers (150 and 200) were obtained by adding 50 (RENUM's third parameter) to the previous line number.

We can also execute RENUM with any one or two of these optional line numbers omitted. If the first parameter is left out, the first new line number generated will be 10. If the second parameter is omitted, the renumbering process will begin with the program's first line. If the third parameter is left out, the amount by which new line numbers will be incremented will be 10. These values are known as **default** values. A default value is that used for a parameter when no other value is supplied by the programmer. The default value is automatically supplied by the BASIC interpreter.

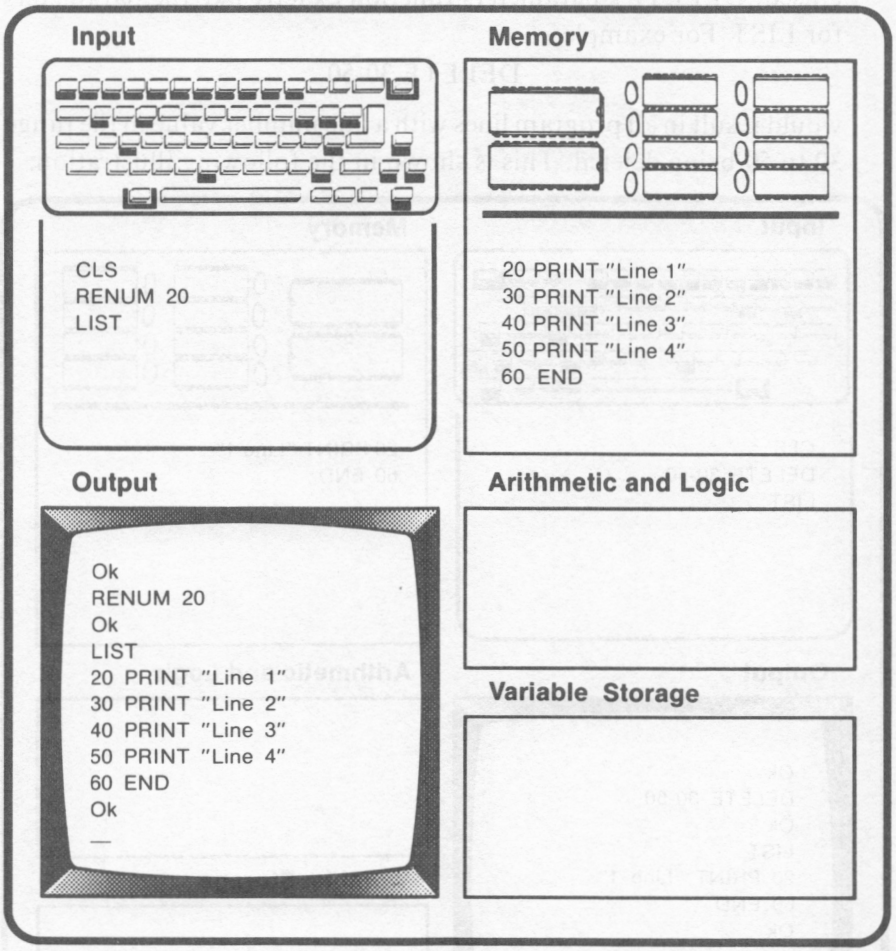
We will use the following example to illustrate the use of default parameter values with RENUM:

```
RENUM ,,20
```



In the preceding RENUM statement, the defaults were used for the first parameter (initial line number value) as well as the second (line number where renumbering is to begin). Therefore, line renumbering began with the program's first line, and the first line number generated was 10. The third parameter (increment for new line numbers) was specified as 20. This is evident from the new line numbers generated (10,30,50,70,90).

Notice that commas were used to denote the places normally occupied by the first and second parameters. If these had not been included, RENUM would have resulted in a different set of line numbers being generated. This is shown in the following illustration:



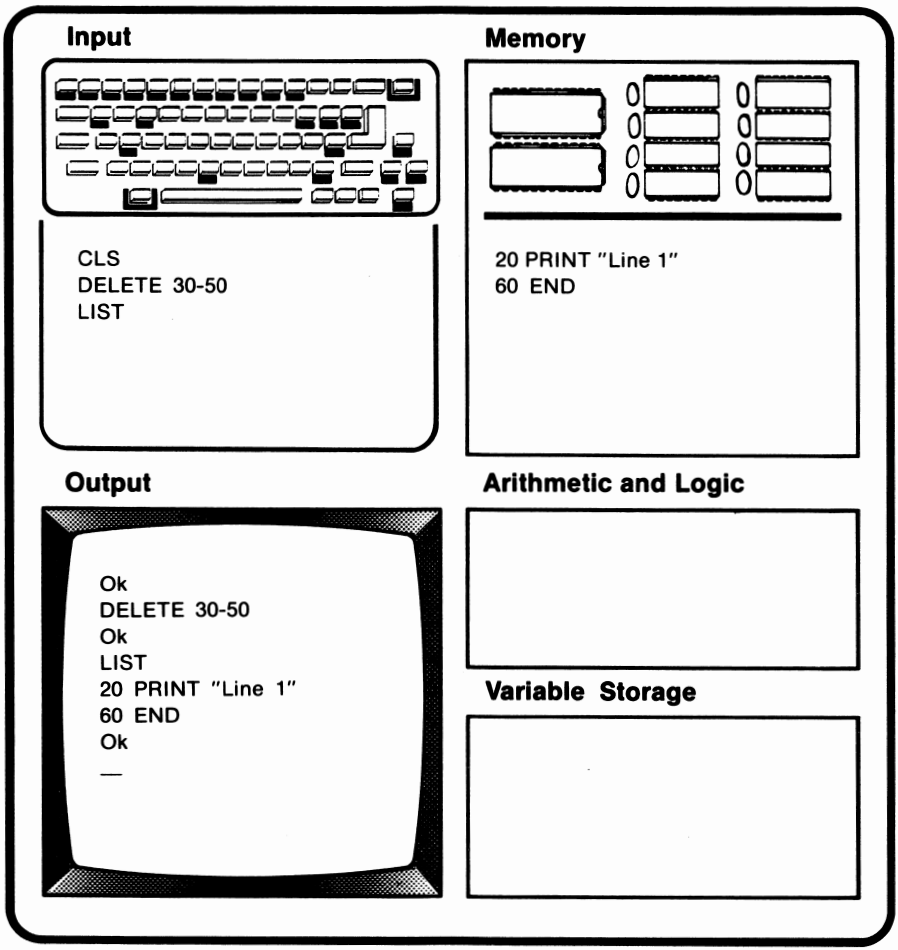
When the commas were omitted, 20 was regarded as RENUM's first parameter. The default values were used for the second and third parameters.

DELETE -- Deleting Program Lines

BASIC's DELETE command is generally used in the immediate mode to erase one or more program lines from memory. DELETE can be used with one or two optional line number parameters. If DELETE is used without any parameters, the entire program will be erased. DELETE's parameters function exactly like the parameters for LIST. For example,

DELETE 30-50

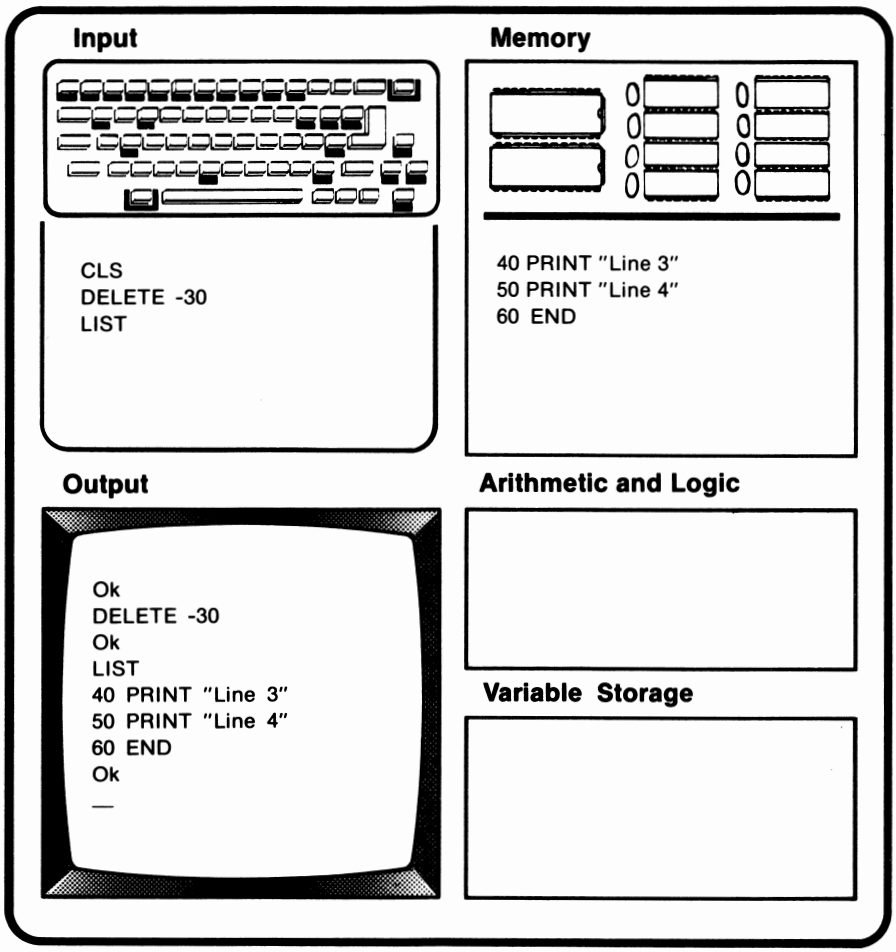
would result in all program lines with a line number value in the range 30 to 50 being deleted. This is shown in the following illustration:



The following statement,

DELETE -30

would cause all program lines from the beginning of the program through line 30 to be erased from memory. This is depicted in the following illustration*:

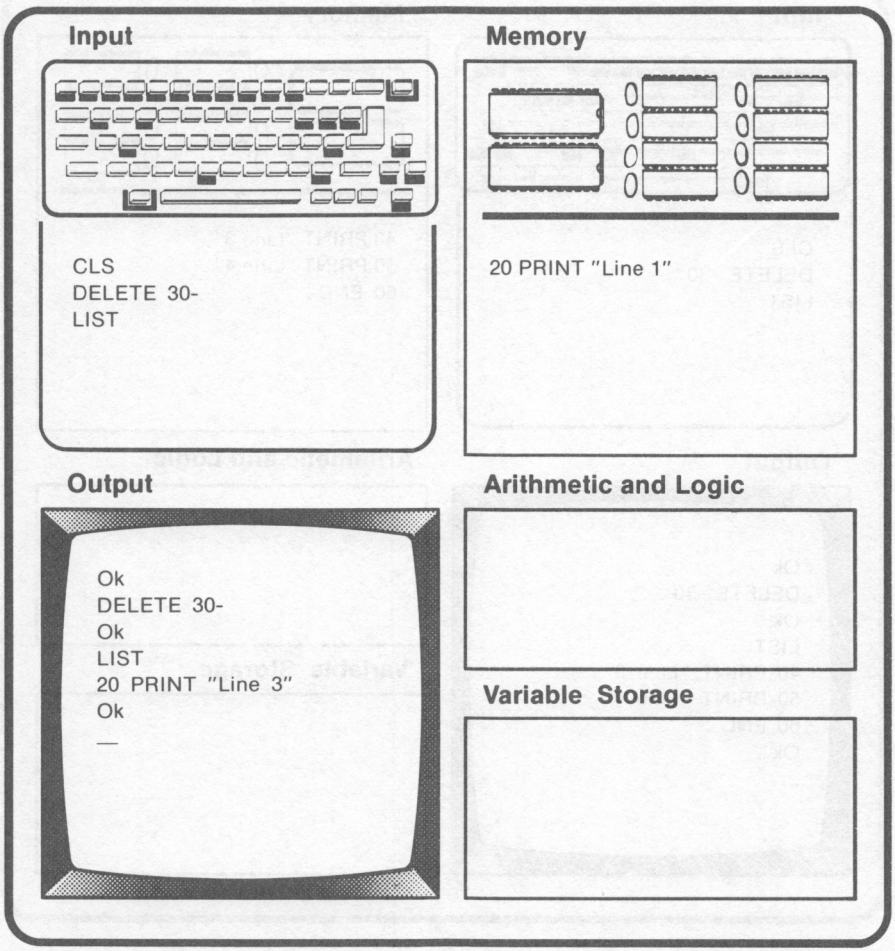


* Assume that lines 30-50 had been restored.

The following statement,

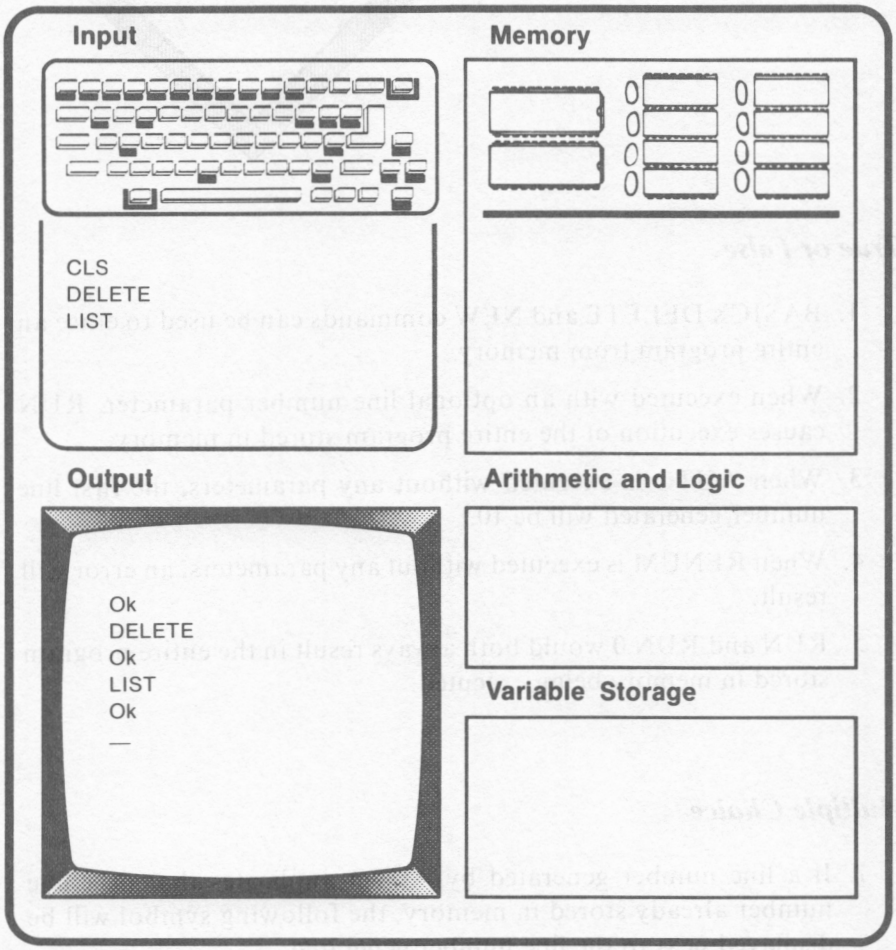
DELETE 30-

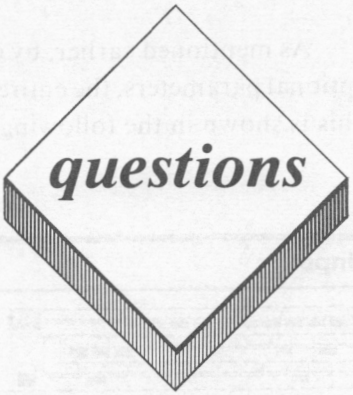
would cause all program lines from line 30 to the end of the program to be erased from memory. The following example illustrates the effect of executing DELETE 30-*:



* Assume that lines 20 and 30 had been restored.

As mentioned earlier, by executing DELETE without any of its optional parameters, the entire program will be erased from memory. This is shown in the following example:





True or False

1. BASIC's DELETE and NEW commands can be used to erase an entire program from memory.
2. When executed with an optional line number parameter, RUN causes execution of the entire program stored in memory.
3. When AUTO is executed without any parameters, the first line number generated will be 10.
4. When RENUM is executed without any parameters, an error will result.
5. RUN and RUN 0 would both always result in the entire program stored in memory being executed.

Multiple Choice

1. If a line number generated by AUTO duplicates that of a line number already stored in memory, the following symbol will be displayed next to the line number generated:

   A. !
   B. ?
   C. *
   D. +
   E. None of the above

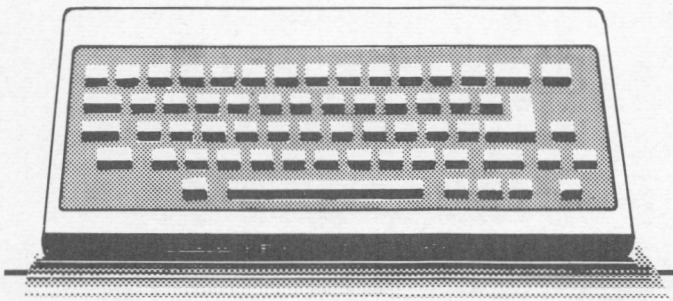
2. The following is not a valid version of the LIST command:
   A. LIST -19
   B. LIST 21-4
   C. LIST
   D. LIST 17
   E. LIST 17-

3. The following key sequence causes the AUTO command to end:
   A. AUTO OFF
   B. NO AUTO
   C. Fn-Break
   D. Break
   E. None of the above

4. The following command can be used to erase a portion of a BASIC program stored in memory:
   A. RUN
   B. LIST
   C. NEW
   D. CLS
   E. None of the above

Computer Exercises

1. Using the program you wrote for Computer Exercise 1 on page 128, perform the following:
   a. Execute RUN so that only those students whose names appear after L in the alphabet are displayed.
   b. Display Chris Matthews' test score.
   c. Erase the program line containing Reid Nagle's test score.
   d. Renumber the program so that the first line number is 1000 and an increment of 50 is used for subsequent line numbers.



Editing Your BASIC Program

lesson 9

Lesson Goals

- *Learn what the BASIC editor is and how it is executed*
- *Learn the function of the various editing keys*
- *Learn how to perform simple editing functions*

Introduction

Once you have entered a BASIC program, you may discover an error that requires that one or more program lines be changed, or **edited.** One means of editing a program line is to simply delete it by entering a corrected line with an identical line number. We will refer to this method as line entry editing.

Another method of editing consists of displaying the program line to be changed using BASIC's EDIT command. We will refer to this method as EDIT command editing.

A final editing method involves listing a program to the screen, positioning the cursor to the point within the line which is to be corrected, making the correction, and sending the corrected line to memory by pressing the Enter key. We will refer to this method as cursor movement editing.

Each of these editing methods will be described in this lesson. We urge you to follow this lesson's examples using your PC*jr*.

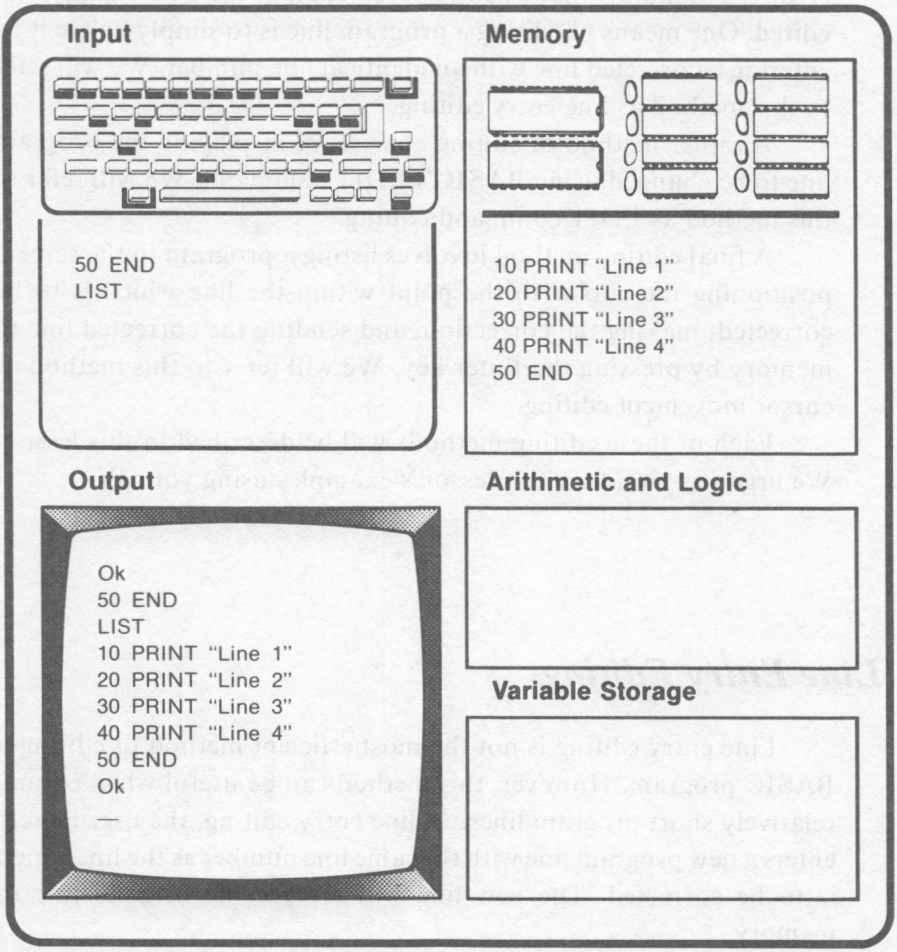
Line Entry Editing

Line entry editing is not the most efficient method of editing a BASIC program. However, this method can be useful when editing relatively short program lines. In line entry editing, the user merely enters a new program line with the same line number as the line which is to be corrected. The new line will replace the original line in memory.

For example, suppose the following program was stored in the PC*jr*'s memory:

```
10 PRINT "Line 1"
20 PRINT "Line 2"
30 PRINT "Line 3"
40 PRINT "Line 4"
50 EMD
```

Obviously, an entry error was made in line 50. "EMD" should have been entered as "END". The following entries would correct this situation:



Editing Keys

Before attempting command editing, you should familiarize yourself with the usage of the various PC*jr* editing keys. These keys and their functions are listed in table 9.1. We will provide specific examples of the usage of the editing keys in the following sections.

Table 9.1. PC*jr* editing keys

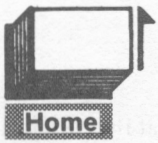
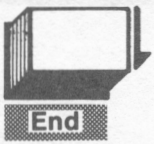
Editing Key	Function
 **Cursor Up**	When the keyboard is in the unshift position, pressing the Cursor Up key causes the cursor to move up one line on the screen.
 **Cursor Down**	When the keyboard is in the unshift position, pressing the Cursor Down key causes the cursor to move down one line on the screen.
 **Cursor Right**	When the keyboard is in the unshift position, pressing the Cursor Right key causes the cursor to move one position to the right. When the cursor reaches the right edge of the screen, pressing the Cursor Right key causes it to move to the farthest left side of the next screen line.
 **Cursor Left**	When the keyboard is in the unshift position, pressing the Cursor Left key causes the cursor to move one position to the left. When the cursor is positioned at the left edge of the screen, pressing the Cursor Left key will cause it to move to the right edge of the preceding line.

Table 9.1 (cont.) PC*jr* editing keys

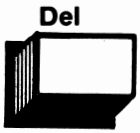
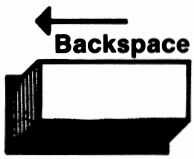
Editing Key	Function
 **Delete**	The Delete key causes the character at the current cursor position to be erased. All characters to the right of the cursor will move one position to the left each time the Delete key is pressed.
 **Insert**	Pressing the Insert key turns on the insert mode. If the Insert key is pressed a second time, the insert mode will be turned off. The insert mode can also be turned off by pressing any of the cursor movement keys or the Enter key. When the insert mode is on, the flashing cursor resembles the following: ■ With the insert mode on, characters can be inserted into an existing line of characters. These characters will be inserted to the immediate left of the flashing cursor. The cursor and all characters following it will be moved one position to the right to make room for the newly inserted characters.
 **Backspace**	The Backspace key deletes the character to the cursor's immediate left. The cursor, the character at the cursor position, and all characters to the right of the cursor will move one position to the left.

Table 9.1 (cont.) PC*jr* editing keys

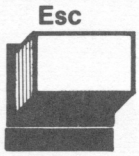
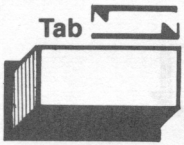
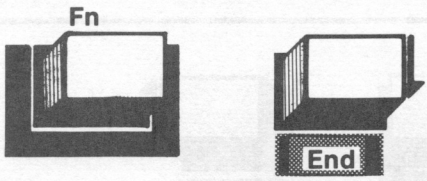
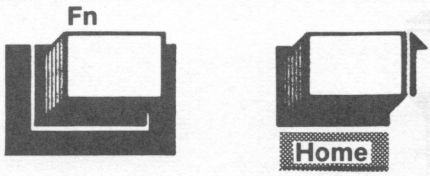
Editing Key	Function
 **Escape**	Pressing the Escape key causes the entire program line in which the cursor is located to be erased from the screen. However, the program line will not be erased from memory.
 **Tab**	The Tab key moves the cursor to the next tab stop. Tab stops are located at every eight character positions on the display line (positions 1,9,17,25, etc.).
 **Fn-End**	The Fn-End key combination moves the cursor to the end of the program line. Any subsequent keyboard entries will be added to the program line.
 **Fn-Home**	The Fn-Home key combination moves the cursor to the screen's home or upper left-hand corner.

Table 9.1 (cont.) PC*jr* editing keys

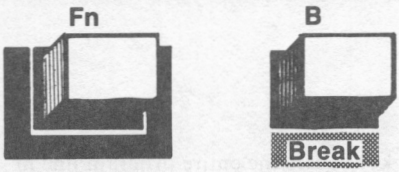
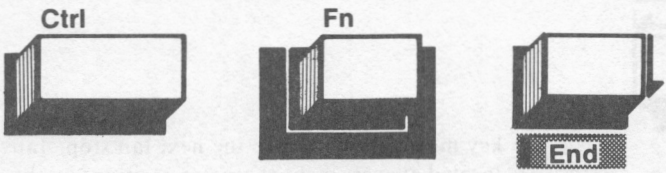
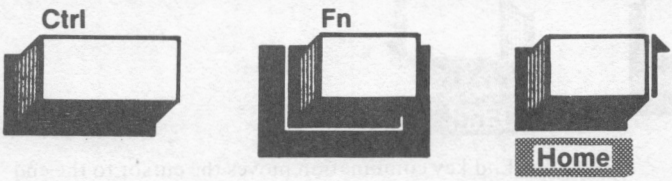
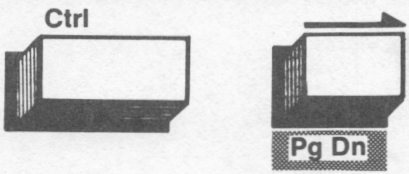
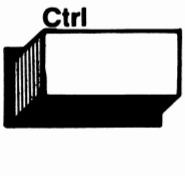
Editing Key	Function
	
Fn-Break	When the edit mode is active, pressing Fn-Break cancels the edit mode. The line being edited will not be saved in memory.
	
Ctrl-Fn-End	The Ctrl-Fn-End key combination causes the program line to be erased from the cursor position to the line's end.
	
Ctrl-Fn-Home	The Ctrl-Fn-Home key combination clears the screen and homes the cursor.
	
Ctrl-Pg Dn	The Ctrl-Pg Dn key combination moves the cursor to the next **word.** A word can be defined as a character or group of characters which begins with a letter or a numbers. Words are separated by blank spaces or special characters.

Table 9.1 (cont.) PC*jr* editing keys

Editing Key	Function
	
Ctrl-Pg Up	The Ctrl-Pg Up key combination moves the cursor to the previous word.

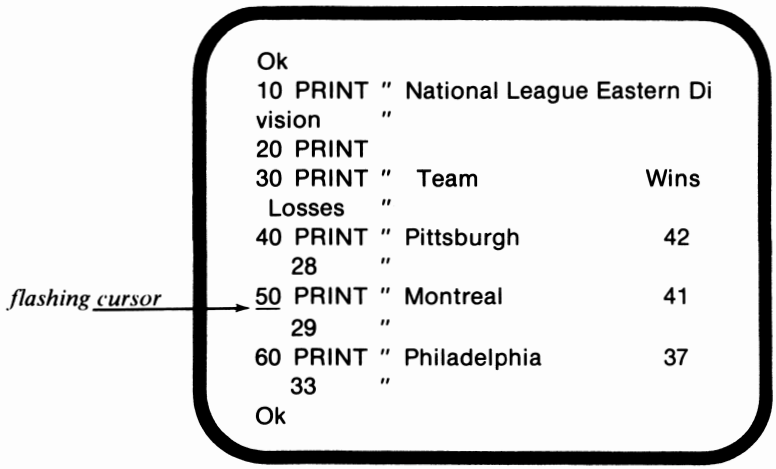
Cursor Up

As defined in table 9.1, pressing the Cursor Up key causes the cursor to move up on the screen as shown below:

Initial screen display

```
Ok
10 PRINT " National League Eastern Di
vision "
20 PRINT
30 PRINT "  Team                Wins
 Losses "
40 PRINT " Pittsburgh            42
   28 "
50 PRINT " Montreal              41
   29 "
60 PRINT " Philadelphia          37
   33 "
Ok
_ ← flashing cursor
```

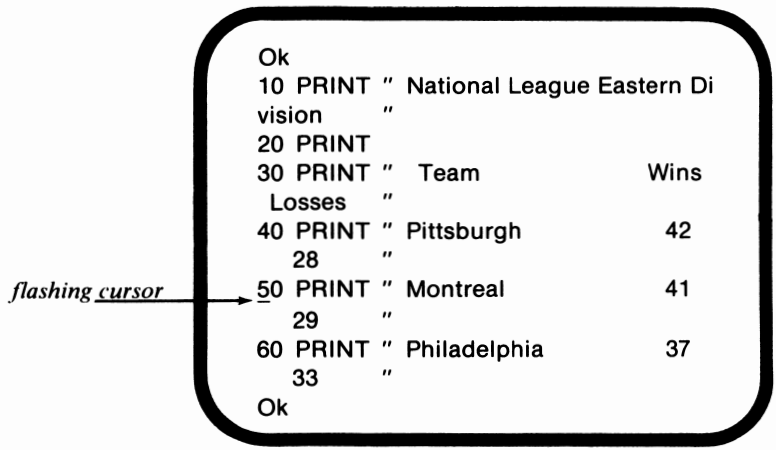
Press ↑ ↑ ↑ ↑ ↑



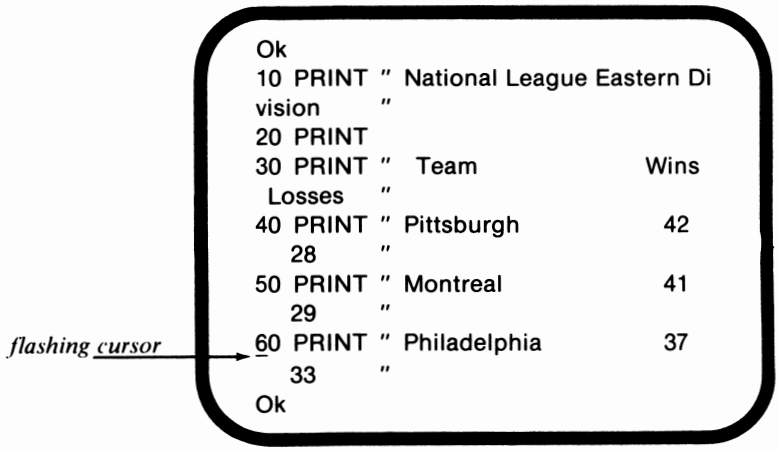
Cursor Down

From table 9.1, we learned that pressing the Cursor Down key causes the cursor to move down one line on the screen. This is shown in the following example:

Initial screen display



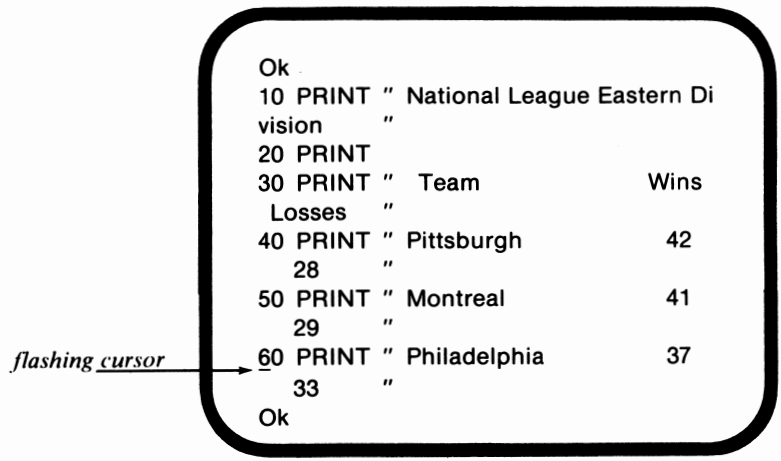
Press ↓ ↓



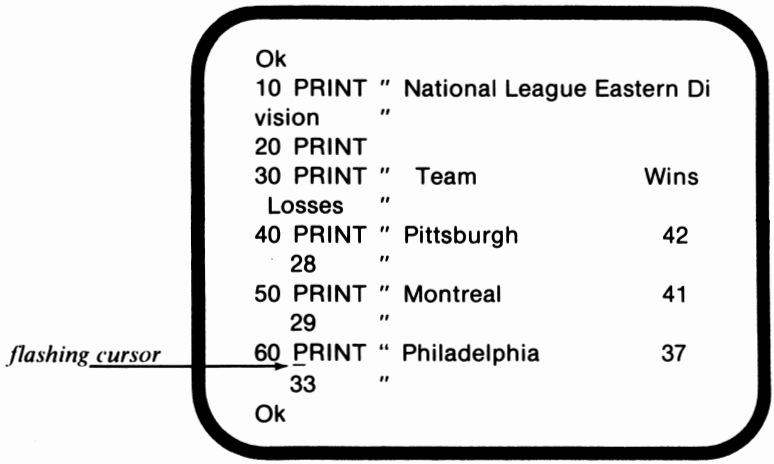
Cursor Right

Pressing the Cursor Right key causes the cursor to move one position to the right on the screen. This is shown in the following examples:

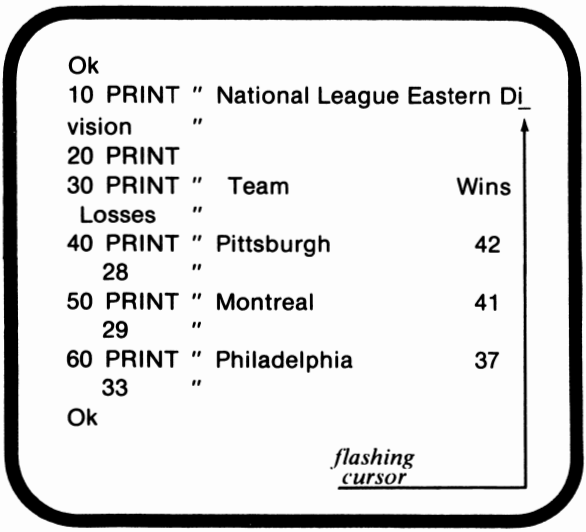
Initial screen display



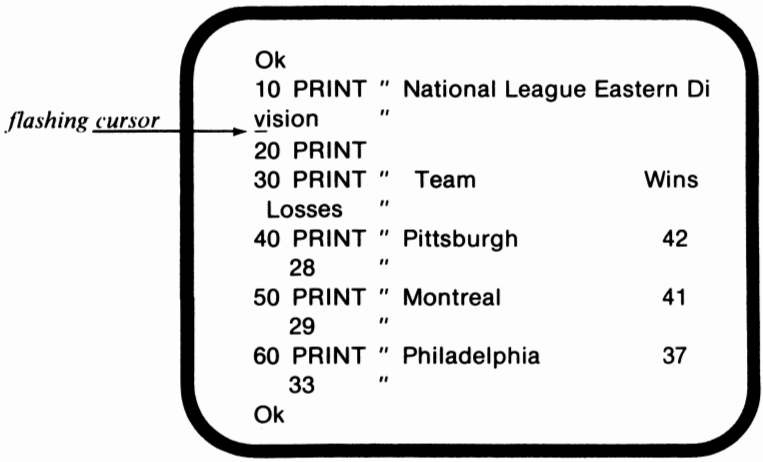
Press → → →



Initial screen display (second example)



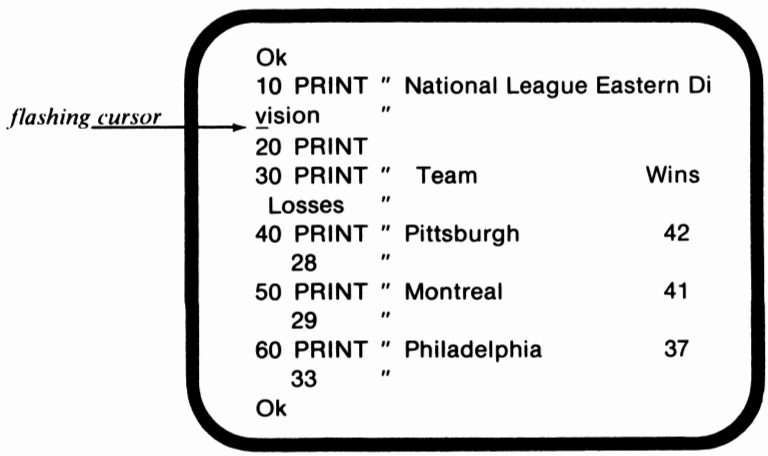
Press →



Cursor Left

Pressing the Cursor Left key causes the cursor to move one position to the left on the screen as shown in the following examples:

Initial screen display



Press ← ←

```
Ok
10 PRINT " National League Eastern Di
vision   "
20 PRINT
30 PRINT "  Team                  Wins
 Losses  "
40 PRINT " Pittsburgh              42
   28    "
50 PRINT " Montreal                41
   29    "
60 PRINT " Philadelphia            37
   33    "
Ok
```

flashing cursor

Delete

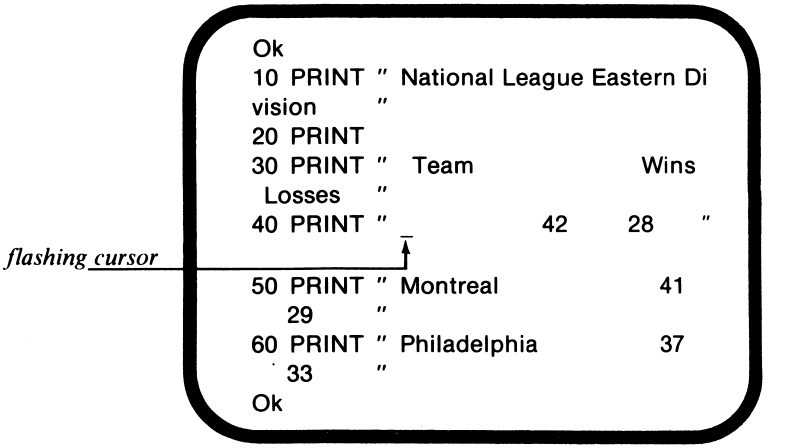
When the Delete key is pressed, the character at the current cursor position will be erased from the screen. All characters to the right of the cursor will move one position to the left when the Delete key is pressed.

Initial screen display

flashing cursor

```
Ok
10 PRINT " National League Eastern Di
vision   "
20 PRINT
30 PRINT "  Team                  Wins
 Losses  "
40 PRINT " Pittsburgh              42
   28    "
50 PRINT " Montreal                41
   29    "
60 PRINT " Philadelphia            37
   33    "
Ok
```

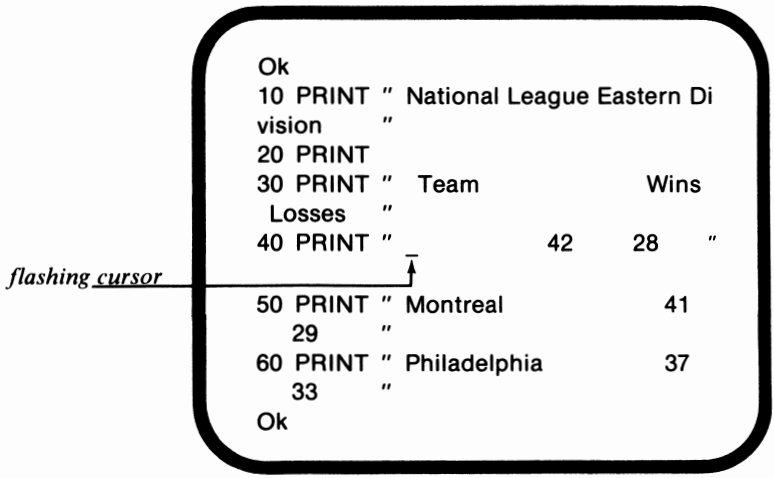
Press Del ten times.



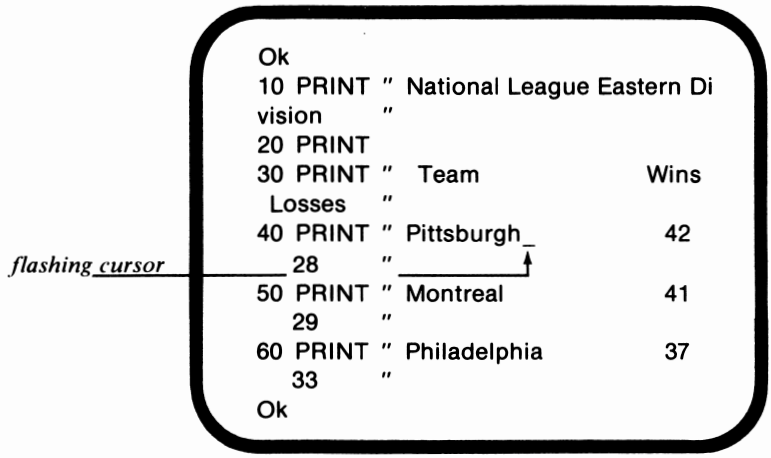
Ins

The Ins key turns on the insert mode. When the insert mode is on, characters can be inserted to the immediate left of the flashing cursor. The cursor and existing characters will be moved one position to the right each time a new character is inserted. The insert mode can be turned off by pressing the Ins key a second time or by pressing any of the cursor movement keys or the Enter key. The following example illustrates the use of the Ins key:

Initial screen display



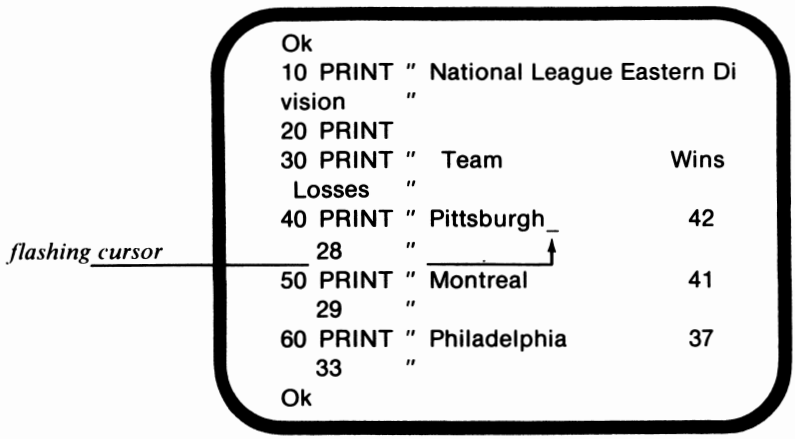
Press Ins Pittsburgh Ins



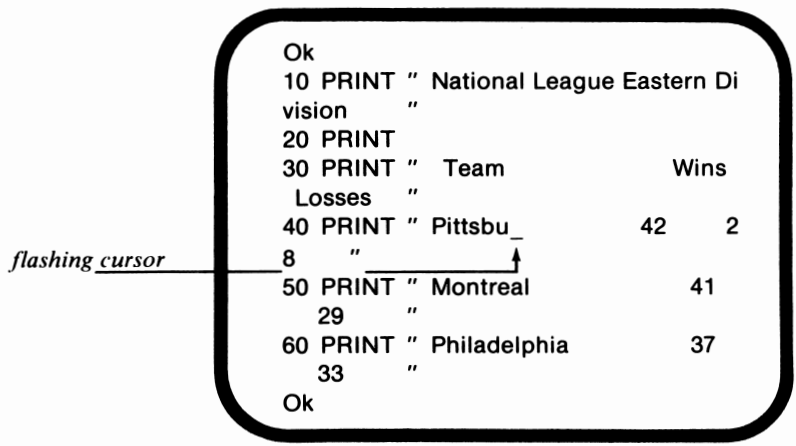
Backspace

The Backspace key erases the character to the immediate left of the cursor. The cursor, the character at the cursor position, and all characters to the left of the cursor will move one position to the left. This is shown in the following example:

Initial screen display



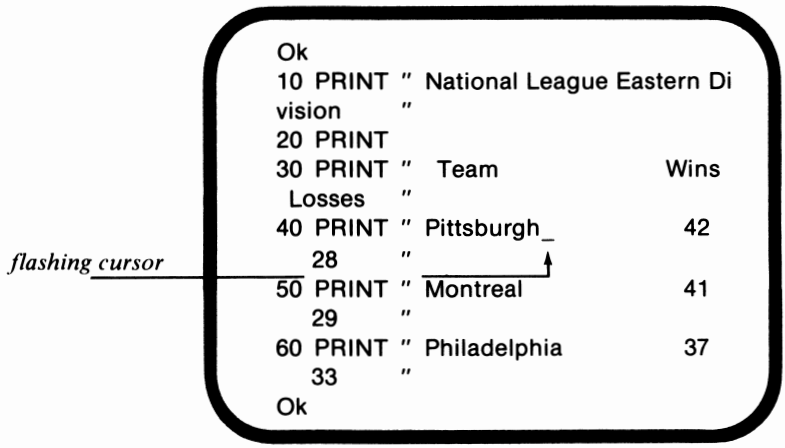
Press Backspace three times



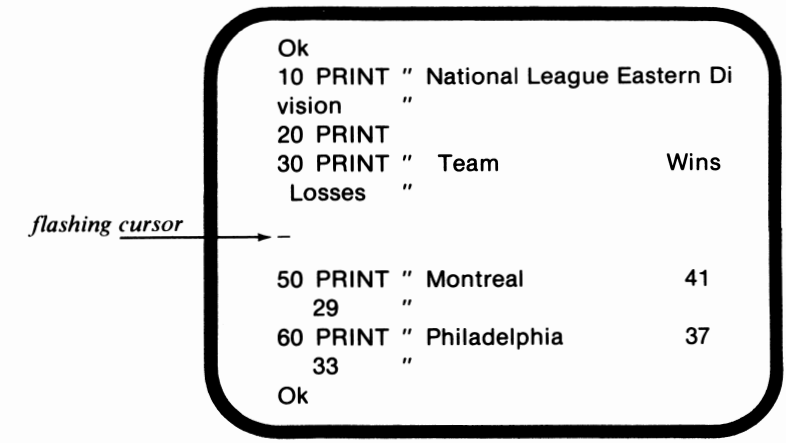
Esc Key

When the Esc key is pressed, the entire program line in which the cursor is located will be erased from the screen. However, that line will not be erased from memory. The following example illustrates the usage of the Esc key:

Initial screen display



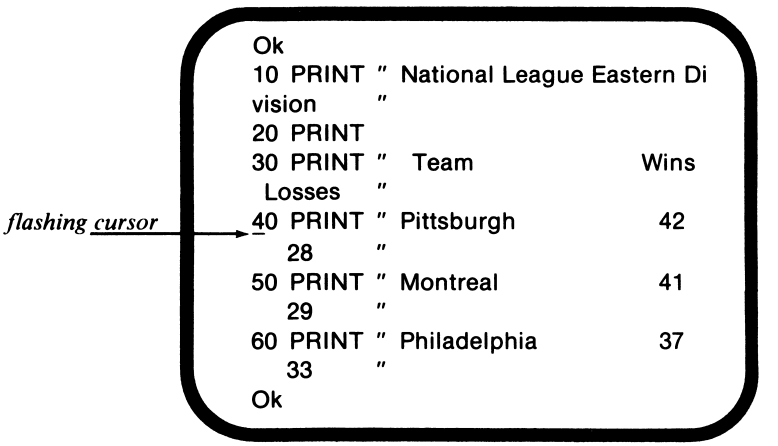
Press Esc



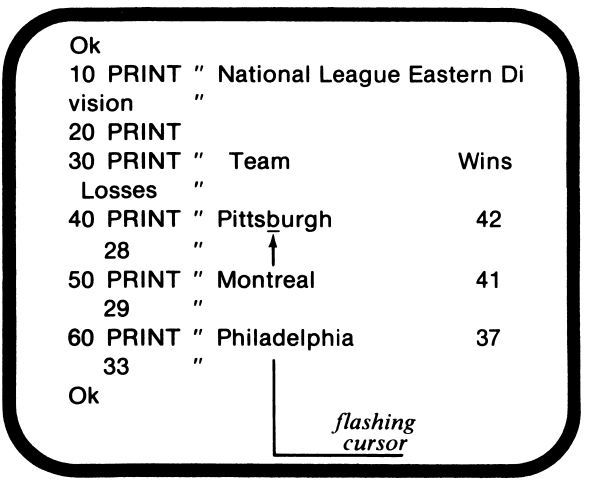
Tab

The Tab key is used to move the cursor to the next tab stop. The effect of using the Tab key with the insert mode off and on is shown in the following examples:

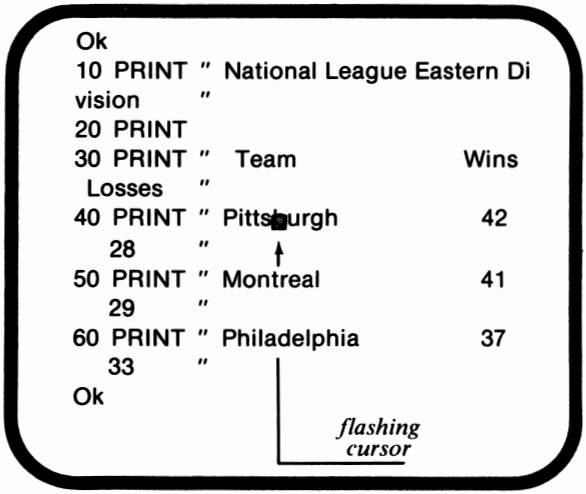
Initial screen display -- insert mode off



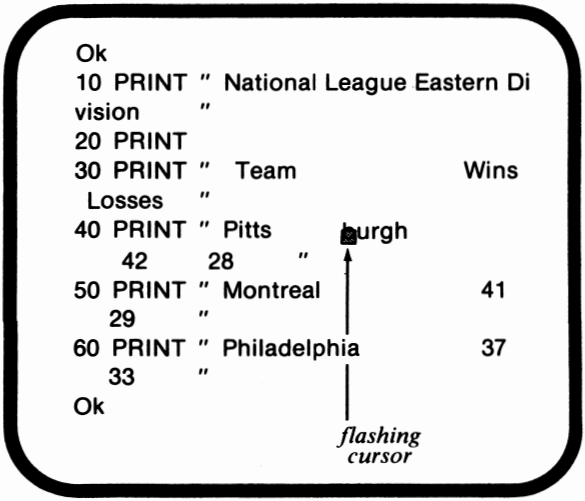
Press Tab twice



Initial screen display -- insert mode on



Press Tab once

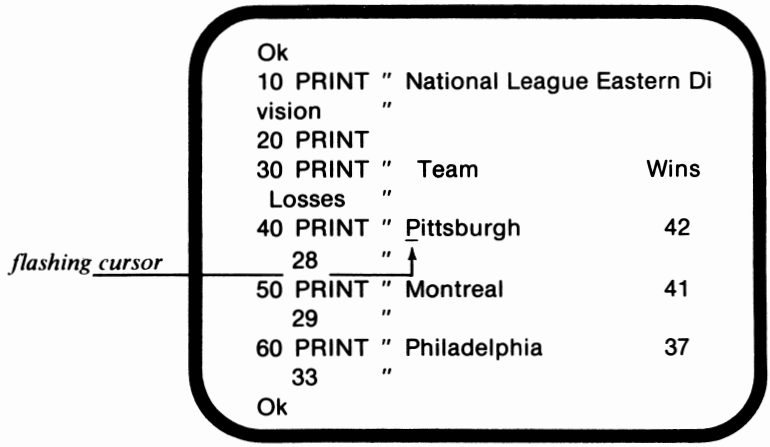


When the insert mode is on, pressing the Tab key causes blank characters to be inserted from the current cursor position to the next tab stop. This was shown in our preceding example.

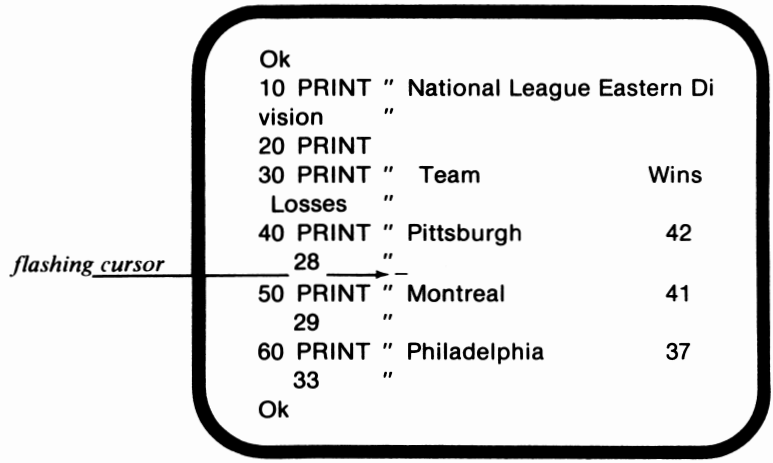
Fn-End

The Fn-End key combination results in the cursor being moved to the end of the current program line. Additional characters can then be added to that line.

Initial screen display



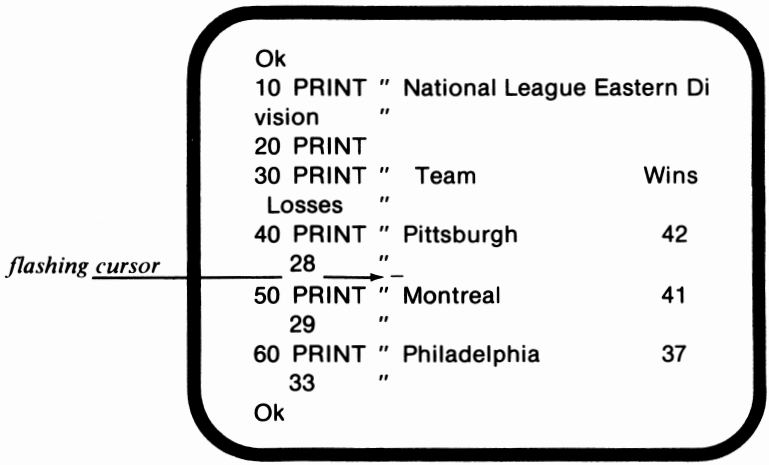
Press Fn-End



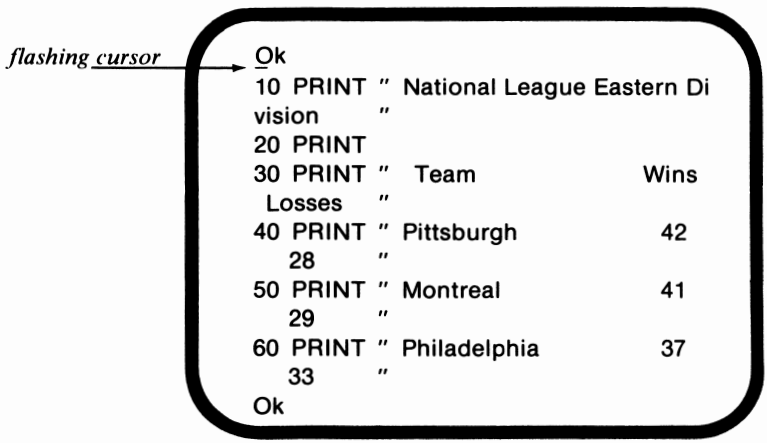
Fn-Home

The Fn-Home key combination moves the cursor to the screen's home or upper left-hand corner. The usage of the Fn-Home key combination is shown in the following example:

Initial screen display



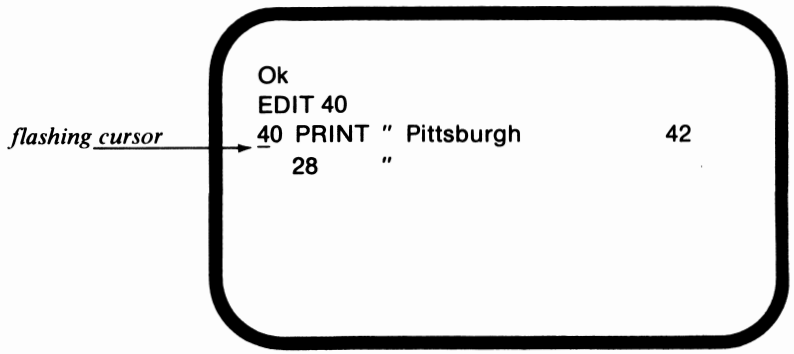
Press Fn-Home



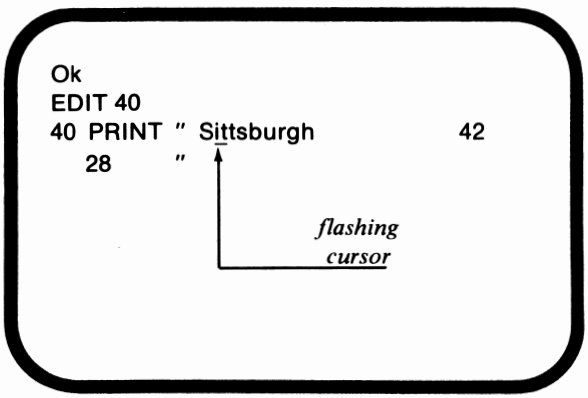
Fn-Break

The Fn-Break key combination is used to cancel the edit mode. The line being edited will not be saved in memory.

Initial screen display



Press → eleven times. Press S



* The EDIT command will be discussed later in this chapter.

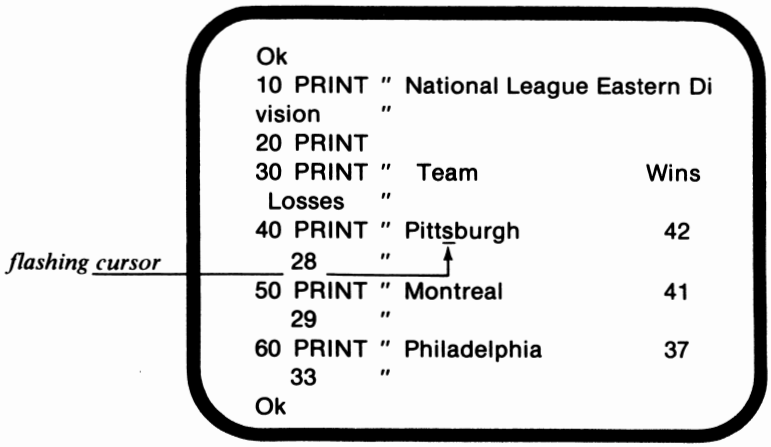
Press Fn-Break. Enter LIST 40

```
Ok
EDIT 40
40 PRINT " Sittsburgh                42
   28      "
LIST 40
40 PRINT " Pittsburgh                42
   28      "
Ok
_
```

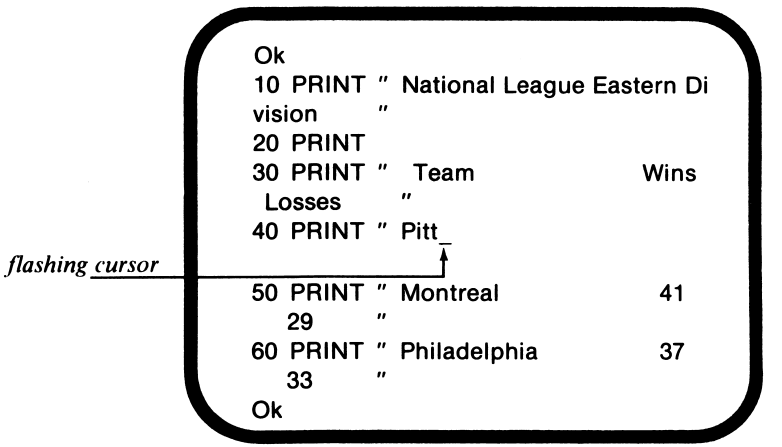
Ctrl-Fn-End

The Ctrl-Fn-End key combination causes the program line to be erased from the current cursor position to the end of the program line.

Initial screen display



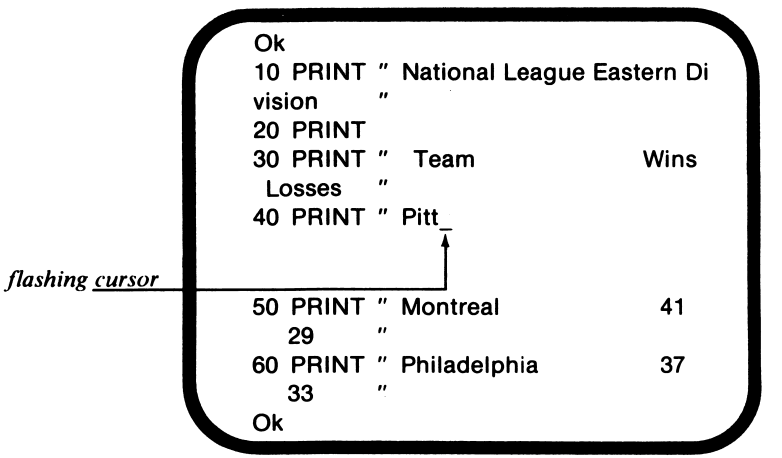
Press Ctrl-Fn-End



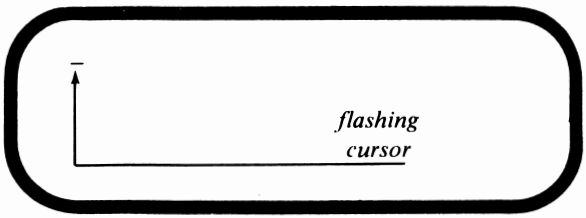
Ctrl-Fn-Home

The Ctrl-Fn-Home key combination clears the screen and returns the cursor to the home position. This is shown in the following example:

Initial screen display



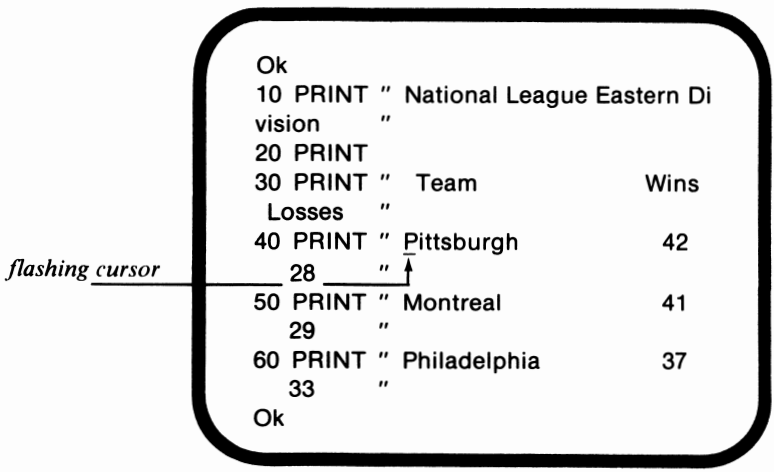
Press Ctrl-Fn-Home



Ctrl-Pg Dn

The Ctrl-Pg Dn key combination causes the cursor to move to the next word. This is shown in the following example:

Initial screen display



Press Ctrl-Pg Dn

```
Ok
10 PRINT " National League Eastern Di
vision     "
20 PRINT
30 PRINT "  Team                  Wins
 Losses    "
40 PRINT " Pittsburgh             42
   28      "                       ↑
50 PRINT " Montreal               41
   29      "
60 PRINT " Philadelphia           37
   33      "                       |
Ok                                 |
                          flashing |
                           cursor  |
```

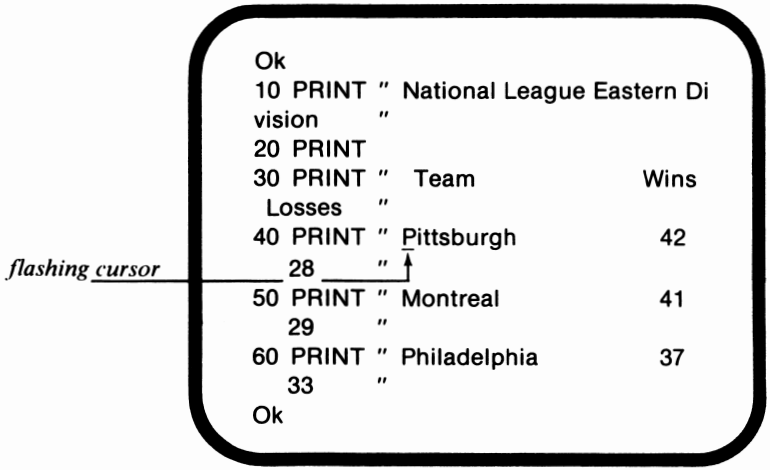
Ctrl-Pg Up

The Ctrl-Pg Up key combination moves the cursor to the previous word. This is shown in the following example:

Initial screen display

```
Ok
10 PRINT " National League Eastern Di
vision     "
20 PRINT
30 PRINT "  Team                  Wins
 Losses    "
40 PRINT " Pittsburgh             42
   28      "                       ↑
50 PRINT " Montreal               41
   29      "
60 PRINT " Philadelphia           37
   33      "                       |
Ok                                 |
                          flashing |
                           cursor  |
```

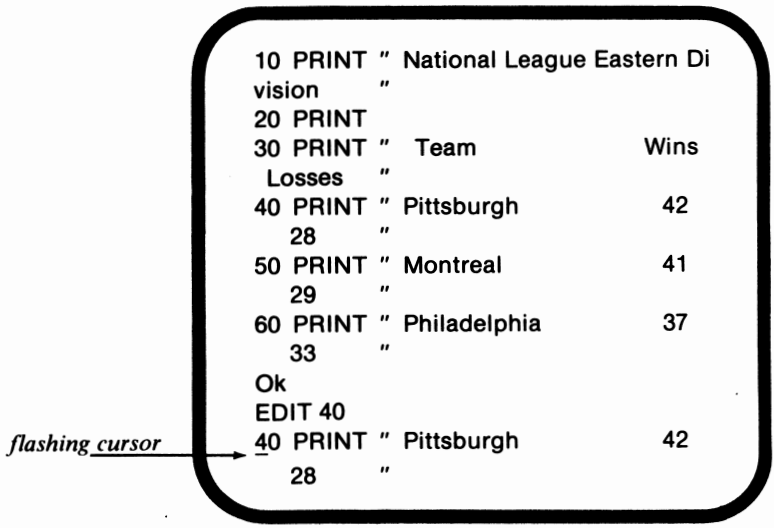
Press Ctrl-Pg Up



EDIT Command Entry

Now that we have learned how the various PC*jr* editing keys function, we are ready to use BASIC's EDIT command to edit program lines. The EDIT command is used to display a BASIC program line so that it can be edited. An example of the use of the EDIT command is given below:

EDIT 40



Notice that the EDIT command must be executed with an existing program line number. The specified program line will then be displayed with the cursor positioned at the first character of its line number.

The program line can then be edited using the various editing keys. For example, suppose that we wanted to change Pittsburgh's number of wins to 43 and its number of losses to 29. We could make these changes by entering the keystrokes indicated in the following example:

Press Tab three times. Press → ten times

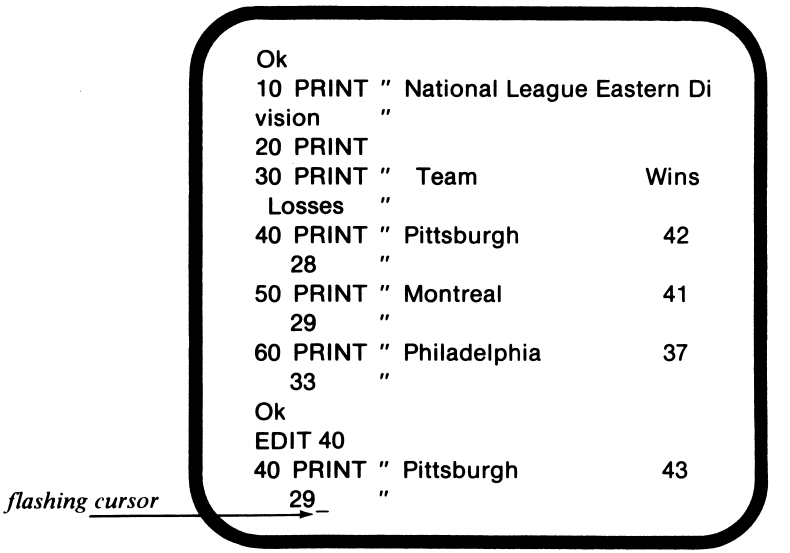
```
10 PRINT "  National League Eastern Di
vision      "
20 PRINT
30 PRINT "   Team                  Wins
 Losses     "
40 PRINT "  Pittsburgh             42
   28       "
50 PRINT "  Montreal               41
   29       "
60 PRINT "  Philadelphia           37
   33       "
Ok
EDIT 40
40 PRINT "  Pittsburgh             42
   28       "
```

flashing cursor

Enter 43.

```
10 PRINT "  National League Eastern Di
vision      "
20 PRINT
30 PRINT "   Team                  Wins
 Losses     "
40 PRINT "  Pittsburgh             42
   28       "
50 PRINT "  Montreal               41
   29       "
60 PRINT "  Philadelphia           37
   33       "
Ok
EDIT 40
40 PRINT "  Pittsburgh             43_
   28       "
```

Press → seven times. Enter 29.



Now that the necessary corrections have been made to line 40, we are ready to send it to memory. We can do so by pressing the Enter key. By executing LIST, we can verify that line 40 has, in fact, been changed.

Cursor Movement Editing

With cursor movement editing, the program is edited by moving the cursor to the desired program line using the cursor control keys; entering the new characters; and pressing the Enter key so as to save the revised line in memory. In the following example, we will use cursor movement editing to change Pittsburgh's number of wins to 43 and its number of losses to 29.

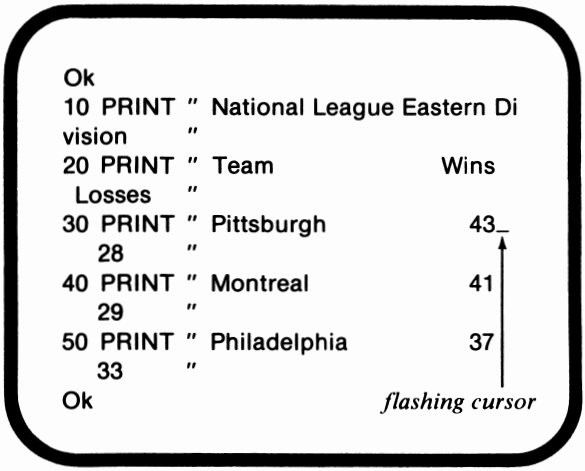
Initial screen display

```
Ok
10 PRINT " National League Eastern Di
vision   "
20 PRINT "  Team                Wins
 Losses   "
30 PRINT " Pittsburgh            42
   28     "
40 PRINT " Montreal              41
   29     "
50 PRINT " Philadelphia          37
   33     "
Ok
_
```

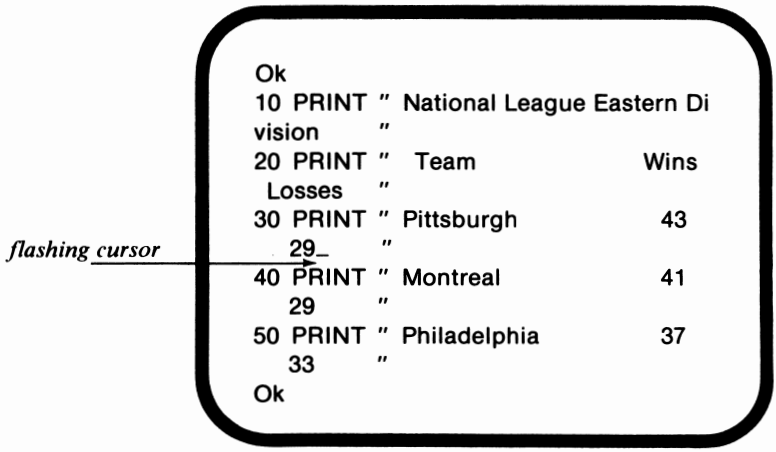
Use → and ↑ to position cursor as indicated

```
Ok
10 PRINT " National League Eastern Di
vision   "
20 PRINT "  Team                Wins
 Losses   "
30 PRINT " Pittsburgh            42
   28     "                      ↑
40 PRINT " Montreal              41
   29     "                      |
50 PRINT " Philadelphia          37
   33     "                      |
Ok                        flashing cursor
```

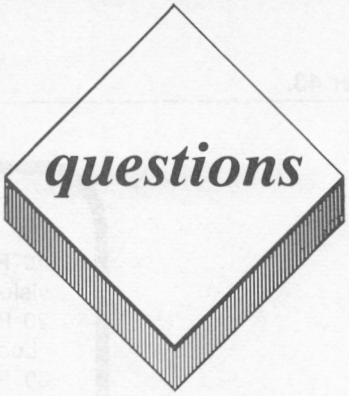
Enter 43.



Press → seven times. Enter 29.



Just as with EDIT, we must send the corrected program line into memory. This can be accomplished by pressing the Enter key.



True or False

1. Line entry editing is the most efficient means of correcting lengthy program lines.
2. When the Home key is pressed, the cursor will move to the home position.
3. BASIC's EDIT command can be used to display a program line for editing.
4. The EDIT mode can be cancelled by pressing the Fn-Break key combination.
5. If the Enter key is not pressed, the changes entered to a program line during a cursor movement editing session will not be saved in memory.

Multiple Choice

1. The following key can be used to delete an entire program line with a single keystroke:
   A. End
   B. Escape
   C. Delete
   D. Backspace
   E. All of the above

2. When pressed, the following key will turn off the insert mode:

   A. Insert
   B. Enter
   C. Cursor Right
   D. Cursor Left
   E. All of the above

3. When pressed, the following key(s) will move the cursor to the next word:

   A. Tab
   B. Ctrl-Pg Up
   C. Cursor Right
   D. Insert
   E. None of the above

4. Which of the following editing methods would be the most efficient means of editing a long program line in the middle of a lengthy program?

   A. Line entry method
   B. EDIT command method
   C. Cursor movement editing
   D. All of the above
   E. None of the above

5. When pressed, the following key will delete characters to the left of the cursor:

   A. Cursor Left
   B. Delete
   C. Backspace
   D. All of the above
   E. None of the above

Computer Exercises

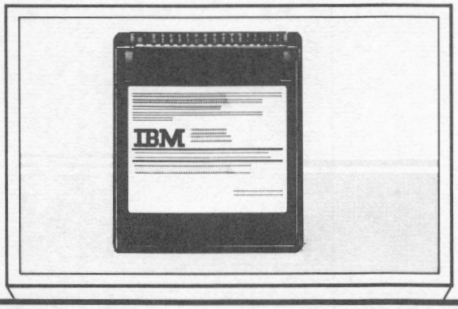
1. Using the program you wrote for Computer Exercise 1 on page 128, perform the following:

   a. Use the EDIT command to change John Croghan's percentage score to 98% and his grade to an A.

b. Delete Reid Nagle's score and grade, leaving only his name and the word "Dropped" where his score previously appeared.

c. Insert the middle name "Ann" in the program line containing Mary Donner's name, score, and grade.

d. Change William Vorhis's first name to Bill.



Saving & Loading BASIC Programs

lesson 10

Goals

- *Learn how to install a cassette recorder with the PCjr*
- *Learn how to save and load a BASIC program using the cassette recorder*
- *Learn how to save and load a BASIC program using the diskette drive*
- *Learn how to use DOS's FORMAT and DISKCOPY commands to format a blank diskette*
- *Learn how to use BASIC's FILES command to display the diskette directory*

Introduction

In the preceding lessons, you have learned how to write and execute BASIC programs. One of the disadvantages you have already encountered in your program writing experience is that once the PC*jr* is turned off, the program in memory will be erased. If you later wish to run that program again, you must reenter it using the keyboard. As you've already learned, this can be a tedious process.

Fortunately, BASIC programs can be stored on either a cassette tape or a diskette for later recall and use. In this chapter, you will learn how to save and recall BASIC programs both on cassette and on diskette.

Cassette Recorder

Before beginning our discussion of the procedure for program storage and retrieval on a cassette tape, we will describe the process for installing a cassette player/recorder with the PC*jr*.

Cassette Recorder Installation

Most standard cassette recorders can be used with the PC*jr*. An optional cassette adapter cable is required to plug the cassette recorder into the PC*jr*. As shown in figure 10.1, the cassette adapter cable has 3 plugs at one end and a single connector at the other end. The single connector should be plugged into the port labeled C on the rear of the PC*jr*'s system unit.

Notice that the three plugs at the other end of the cable are colored black, red, and grey. These plugs are connected to the cassette recorder. The black plug should be connected to the recorder's earphone (or monitor) jack. The red plug should be connected to the auxiliary or microphone jack.

If your cassette recorder has a jack labeled rem, the grey plug can

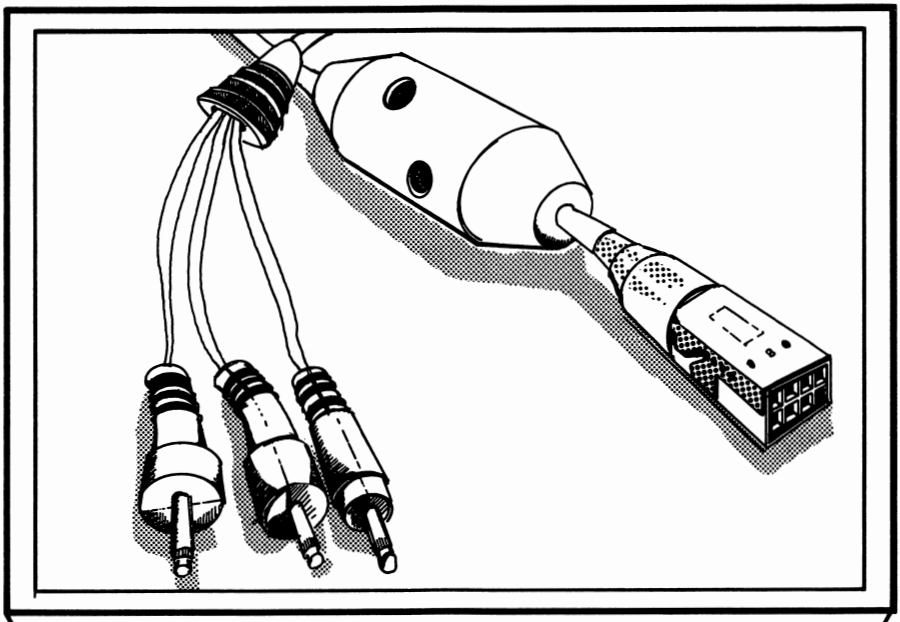


Figure 10.1. PC*jr* cassette adapter cable

be installed in that port. This connection is used for automatic operation of the cassette recorder. If your recorder does not have a rem port, leave the grey plug disconnected. Since most cassette recorders do not have this port, our discussion of cassette recorder usage will be based on the assumption that the grey plug has been left disconnected. Therefore, at this point, please leave the grey plug disconnected.

Your final installation step is to plug the cassette recorder's power cord into a wall outlet. Although your recorder may operate with batteries, we recommend against using battery power when saving and loading programs. As the batteries run down, the recorder will run more slowly. When new batteries are installed, the PC*jr* may not be able to load those programs recorded at the slower speed.

The correct cassette recorder installation is depicted in figure 10.2. You can either test your installation by operating the cassette recorder with a blank cassette installed or by operating it without a

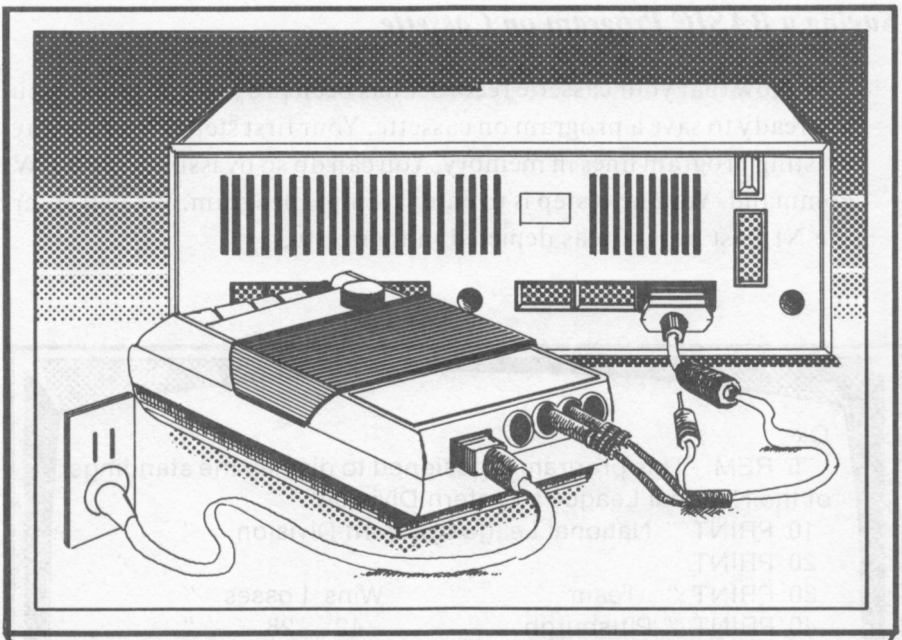


Figure 10.2. Cassette recorder installation

cassette. Press the recorder's play button. The right-hand spindle should rotate. Then, press the stop button. The spindle should stop rotating. Press the recorder's rewind button. The left-hand shaft should rotate. Again, when the stop button is pressed, the spindle should stop rotating.

If your cassette recorder does not operate properly, check the recorder's power cord to be certain that it is connected properly to both the cassette unit and to the wall outlet. Also, check the black and red plugs to be certain that they are properly installed. The grey plug should not be connected to the recorder.

One final cassette recorder feature that you should be aware of is the volume control. The volume control must be set at a sufficient level to enable data to be recorded and later be played back. Generally, a volume setting in the middle of the allowed range enables data to be properly recorded. However, if you experience difficulties saving and loading data with the cassette recorder, you may have to adjust the volume control.

Saving a BASIC Program on Cassette

Now that your cassette recorder has been properly installed, you are ready to save a program on cassette. Your first step is to erase any existing program lines in memory. You can do so by issuing the NEW command. Your next step is to enter a simple program. We will enter the NLEast program as depicted in figure 10-3.

```
Ok
  5 REM "This program is designed to display the standings
of the National League's Eastern Division"
 10 PRINT "  National League Eastern Division     "
 20 PRINT
 30 PRINT "   Team                 Wins  Losses   "
 40 PRINT "  Pittsburgh             42    28      "
 50 PRINT "  Montreal               41    29      "
 60 PRINT "  Philadelphia           37    33      "
 70 PRINT "  St. Louis              34    34      "
 80 PRINT "  Chicago                29    41      "
 90 PRINT "  New York               28    42      "
100 END
```

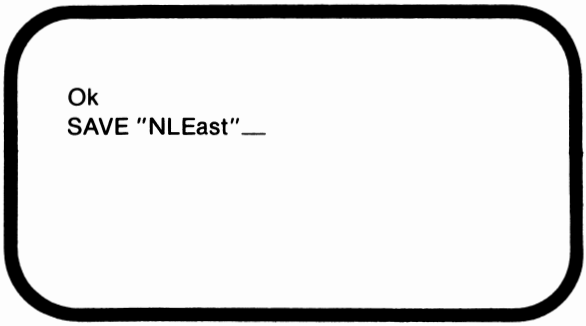
Figure 10.3. NLEast

Once "NLEast" has been properly entered, we can save a copy of it on cassette tape. First of all, you must prepare the cassette recorder by:

- ☐ Inserting a cassette tape
- ☐ Closing the door
- ☐ Pressing the recorder's rewind button
- ☐ After the tape has been completely rewound, press the stop button

Then, make the following keyboard entry if your PC*jr* does not have a diskette drive and cartridge BASIC installed:

SAVE "NLEast" *(Do not press the Enter key)*



Next, press the recorder's play and record buttons simultaneously. Both keys should remain depressed. After the recorder has run for about seven seconds, press the PC*jr*'s Enter key. This will start the recording process. The flashing cursor will disappear from the screen as the program is being recorded. Once the program has been recorded, the Ok prompt and the flashing cursor will reappear. At this point, press the recorder's stop button.

You may have wondered why we allowed the recorder to run seven seconds before pressing the Enter key. Cassette tapes have a section at their beginning on which data cannot be recorded. This area is known as the tape leader. We allowed the recorder to run seven seconds prior to beginning the program saving process in order to skip over the tape leader.

If your PC*jr* has a disk drive and cartridge BASIC installed, you should follow the same steps with one exception. The SAVE command should be entered as follows:

SAVE "CAS1:NLEast"

```
Ok
SAVE "CAS1:NLEast"
```

CAS1: tells the PC*jr* to save NLEast on the cassette recorder rather than the diskette drive. If CAS1: were not included, the program would have been saved on the diskette drive.

Loading a BASIC Program from Cassette

Now that we have saved a copy of "NLEast" on cassette, let's see if it still exists in memory.

LIST

```
Ok
SAVE "NLEast"
Ok
LIST
5 REM "This program is designed to display the
standings of the National League's Eastern Division"
10 PRINT "    National League Eastern Division    "
20 PRINT
30 PRINT "  Team              Wins  Losses    "
40 PRINT " Pittsburgh          42    28       "
50 PRINT " Montreal            41    29       "
60 PRINT " Philadelphia        37    33       "
70 PRINT " St. Louis           34    34       "
80 PRINT " Chicago             29    41       "
90 PRINT " New York            28    42       "
```

Before loading "NLEast" from the cassette tape back into memory, we will have to erase the original version of the program from memory. This can be accomplished by executing the NEW command. If you execute LIST a second time, you will find that the PC*jr*'s memory is now empty.

```
NEW
LIST
```

```
Ok
NEW
Ok
LIST
Ok
_
```

We are now ready to load NLEast from cassette into the PC*jr*'s memory. First of all, be certain that the cassette containing the copy of NLEast is installed in the cassette recorder. The tape should have been rewound to a point prior to the beginning of NLEast. Then, enter the following:

```
LOAD "NLEast"
LOAD "CAS1:NLEast" (if your PCjr has a diskette drive
                    and Cartridge BASIC installed)
```

```
Ok
LOAD "NLEast"
Ok
_
```

This time, press the Enter key immediately after keying in the LOAD command. After pressing the Enter key, press the recorder's play button. The flashing cursor will be absent from the screen as NLEast is searched for and loaded from the cassette tape. Once NLEast has been loaded, the following message will appear on the screen:

```
NLEast found
```

The cursor and the Ok prompt will appear in a few more seconds. At this point, press the cassette recorder's stop button. By executing the LIST command, you will be able to verify that NLEast has in fact been loaded from the cassette tape into memory.

Disk Drive

Before learning the procedure for saving and loading a program on a diskette, you must learn how to format a diskette and how to check a diskette's directory. These will be covered in the following sections.

Formatting a Diskette

Before using a new diskette, you must first format it using DOS's FORMAT command. The formatting process involves recording a specific magnetic pattern on the diskette surface that enables the computer to read and write data.

At this point, let's format a blank diskette. Our first step is to restart the PC*jr* if it is off, or perform a warm boot if it is powered on. After the system date and time have been entered, the DOS prompt (A>) will appear.

To this point, we have been entering BASIC commands such as PRINT, NEW, LIST, and RUN in response to BASIC's Ok prompt. FORMAT, the command for formatting a blank diskette, is a DOS command and must be entered when the DOS prompt is active. Enter FORMAT, and press the Enter key. The following message should appear on the display screen:

```
Insert new diskette for Drive A:
and strike any key when ready
```

Remove the DOS diskette from the drive, and replace it with either a blank diskette or a diskette containing data that can be erased. Once the diskette to be formatted has been installed, press the Enter key. The following messages will appear on the screen. Be patient as the formatting process requires about 60 seconds.

```
Formatting... Format complete

    362496  bytes total disk space
    362496  bytes available on disk

Format another (Y/N)?
```

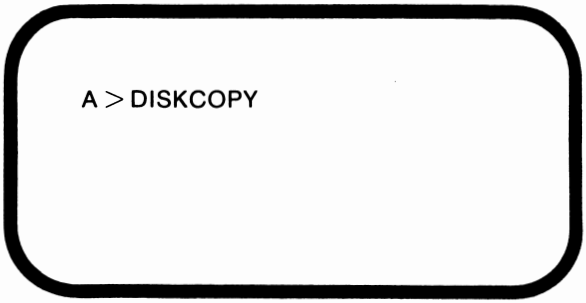
The diskette has now been formatted. If you wish to format another diskette, press Y, and the formatting process will be repeated. Otherwise, press N, and the DOS prompt will reappear.

DOS 2.1 offers a second means of formatting a diskette with its DISKCOPY command. DISKCOPY causes the contents of one diskette, known as the **source diskette,** to be copied onto another diskette, known as the **target diskette.** The advantage to DISKCOPY is that it allows the user to place a copy of DOS on the target diskette while it is being formatted. This can be accomplished by using the DOS 2.1 system master diskette as the source diskette.

Since the PC*jr* contains only one diskette drive, it will be necessary to periodically remove the source diskette and replace it with the target diskette. This process is known as **swapping.** Once DISKCOPY has been executed, you may have to swap diskettes as many as four or five times before the entire copying operation has been completed.

To begin DISKCOPY, enter the following after the DOS prompt:

DISKCOPY



DOS will prompt you as follows when it is necessary to swap diskettes:

```
Insert source diskette in drive A:
Insert target diskette in drive A:
```

When the disk swapping process has been completed, the following message will appear:

```
Copy complete
Copy another (Y/N)?
```

If N is entered, DISKCOPY will end and the DOS prompt will appear.

You can determine whether or not DOS was copied to the target diskette by attempting to perform a warm boot using that diskette. If the system successfully boots, DOS was copied to the target diskette.

Displaying the Diskette Directory

A diskette can contain a number of different programs or data files. Each file generates a listing in the diskette's **directory.** A diskette directory can be defined as a file on the diskette containing information relating to all other files stored on that diskette.

BASIC's FILES command can be used to list the directory entries for all of the files on a diskette. This use of FILES is illustrated below:

FILES

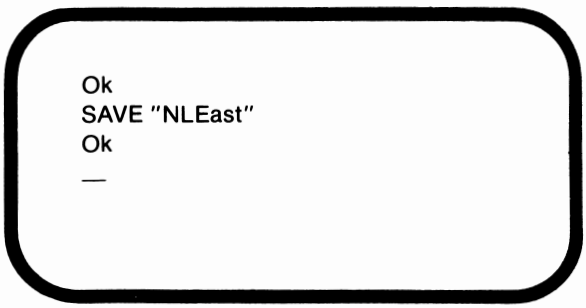
```
Ok
FILES
A:
SORT      .EXE    FIND      .EXE
MORE      .COM    BASIC     .COM
BASICA    .COM    PAINTGRA  .BAS
R                 HARRY#3   .BAS
PAY       .DAT    TEXT
PAY       .DAT    FILE      .DAT
PAY23     .DAT    TEXT      .DAT
JOHN      .BAT    PAY       .DAT
PKLING    .BAS    PK        .BAS
PAT       .BAS    BARACADE  .BAS
FIG6-2    .BAS    JUNK      .BAS
PKL       .BAS    GOODSTUF  .BAS
PJK       .BAS    TEM       .BAS
TEMP      .BAS    ROWBOAT   .BAS
BARA      .BAS    BARA1     .BAS
PJKL      .BAS    PROB1A    .BAS
OK        .BAS    NLEAST    .BAS
9-11      .BAS    9-12      .BAS
9-13      .BAS    6-20      .BAS
```

Saving and Loading BASIC Programs on Diskette

The procedures used to save and load BASIC programs on a diskette are similar to those used on a cassette -- only easier and faster. Once the program to be saved is ready in the PC*jr*'s memory, remove the DOS system master diskette from the PC*jr*'s diskette drive, and replace it with a blank formatted diskette. If you are using a copy of the DOS diskette, programs can be saved directly on the copy -- as long as sufficient room is available.

Suppose NLEast is already present in memory. If not, it can be loaded from a cassette tape as described earlier. Enter the following command:

```
SAVE "NLEast"
```



By executing the FILES command, we can examine the directory to determine whether NLEast has been saved on the diskette. If the preceding command was properly executed, NLEast should appear in the directory listing.

The LOAD command is used to load a program file stored on diskette back into the PC*jr*'s memory. Before executing LOAD, execute the NEW and CLS commands to clear the PC*jr*'s memory and screen, respectively. Next, execute the LOAD command as follows:

LOAD "NLEast"



By executing the LIST command, we can verify that NLEast has been loaded into memory.

Erasing a File from a Diskette

BASIC's KILL command can be used to delete a diskette file. The file whose filename corresponds to that indicated with KILL will be deleted. This is shown in the following example:

KILL



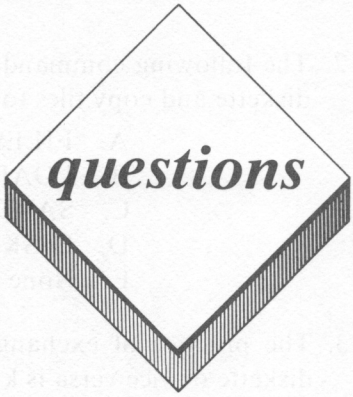
If FILES is subsequently executed, the directory listing will not include an entry for NLEast.

Notice that the filename specified with the KILL command was NLEast.BAS not NLEast. The Microsoft BASIC interpreter requires that BASIC program files include .BAS as their filename extension. You might already have noticed from the directory listings that NLEast was displayed as NLEast.BAS in those listings, even though NLEast had been specified in the SAVE command. When a BASIC program file is saved or loaded, the filename extension .BAS is automatically added. It does not have to be specified with the command. Therefore, the following command,

```
SAVE "NLEast"
```

was interpreted by the compiler as:

```
SAVE "NLEast.BAS"
```



True or False

1. The following command could be used to save a BASIC program on a diskette in a PC*jr* enhanced model's disk drive:

```
SAVE "PROGRAMA"
```

2. If you are saving a program on a new cassette, you can immediately press the Enter key once the SAVE command has been entered and the cassette recorder's play button pressed.
3. A blank diskette must be formatted before information can be copied to it.
4. Either DOS's FORMAT or BASIC's FILES command can be used to format a blank diskette.
5. A directory is a map to the PC*jr*'s internal memory addresses.

Multiple Choice

1. Which of the following BASIC commands is generally used to display a diskette's directory:

   A. DISKCOPY
   B. LIST
   C. FORMAT
   D. LOAD
   E. FILES

2. The following command would be used to both format a blank diskette and copy files to it:

   A. FILES
   B. LOAD
   C. SAVE
   D. DISKCOPY
   E. None of the above

3. The practice of exchanging the source diskette for the target diskette or vice versa is known as:

   A. Disk exchange
   B. Booting
   C. Formatting
   D. Swapping
   E. None of the above

4. Which of the following commands could be used with a PC*jr* with a diskette drive to load a BASIC program named PROGRAMB from a cassette tape?

   A. LOAD PROGRAMB
   B. LOAD "PROGRAMB"
   C. LOAD
   D. All of the above
   E. None of the above

5. The .BAS filename extension must be specified when a BASIC program filename is used with the following BASIC command:

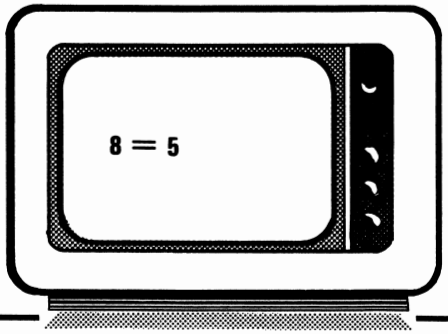
   A. LOAD
   B. SAVE
   C. KILL
   D. All of the above
   E. None of the above

Essay

1. Define the term formatting as it relates to diskettes.
2. Define the term directory as it relates to diskettes.

Computer Exercise

1. Format a blank diskette using FORMAT. Format a second and copy DOS to it using DISKCOPY.
2. Reenter the program you wrote for the exercise on page 128. Save this program on diskette and on cassette. Erase the program from memory and then load it from diskette back into memory. Repeat this procedure using the cassette recorder.
3. Save the program again on diskette using a different filename. Erase this second copy from the diskette.



Data Types and Variables in BASIC

lesson 11

Goals

- *Define string data and use string constants within the context of a simple BASIC program*
- *Understand how ASCII codes are used to convey string data*
- *Learn the various ways in which numeric data can be represented*
- *Define the term variable in the context of BASIC programming*
- *Learn how to create variable names*
- *Learn how BASIC's LET statement can be used to assign a value to a variable*

Introduction

Data can be defined as information that is to be processed by the computer. Information might consist of letters of the alphabet, numbers, or special symbols such as !, ?, @, or *. In this chapter, we will discuss the two types of data used with Microsoft BASIC programs, string and numeric. String and numeric data are stored differently in memory by the PC*jr*.

String Data

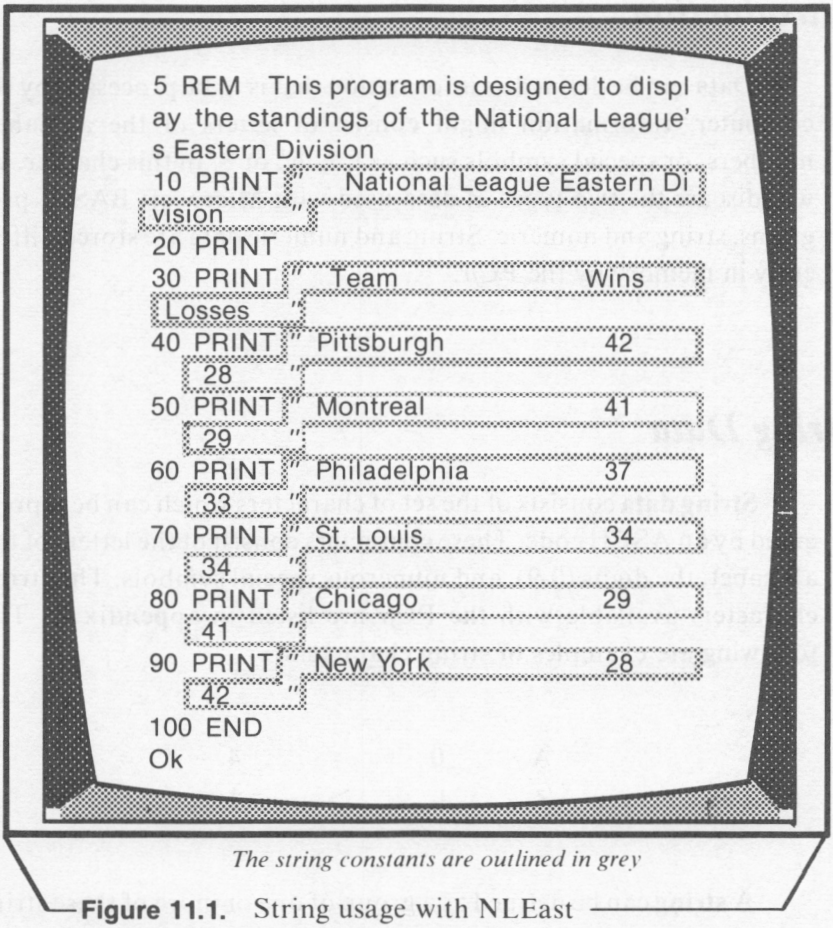
String data consists of the set of characters which can be represented by an ASCII code. These characters consist of the letters of the alphabet, the digits (0-9), and numerous special symbols. The string characters available with the PC*jr* are listed in appendix A. The following are examples of string characters:

A	0	=	4
Z	!	>	?

A **string** can be defined as a group of one or more of these string characters.

String Data Examples

We have already encountered string data in the programs we have written so far in this book. For example, as shown in figure 11.1, NLEast uses a number of different strings.



The string constants are outlined in grey

Figure 11.1. String usage with NLEast

Notice that when a string is used in a PRINT statement, it must be enclosed in quotation marks. This is also true when strings are used with other BASIC statements. A string enclosed within quotation marks is referred to as a **string constant.** A constant can be defined as a data item with a fixed value. The following are examples of string constants:

"Terry Johnson"
"131 Elm Street"
"Charlotte, NC"
"28213"

Notice that numbers can be used as string constants. Keep in mind that when numbers are used as string constants, these numbers cannot be used as numeric data. The numbers are string data and must be used as any other string data item would be.

One final point that should be kept in mind regarding string constants is that they cannot contain quotation marks. For example, the following string constant,

"Bill said, "Goodbye," as he walked away."

would be illegal. Since quotation marks are used to denote the beginning and ending points of a string constant, their inclusion within the string itself would cause difficulties and therefore is not allowed.

In lesson 18, we will discuss how the CHR$ function can be used to place the ASCII code for quotation marks within a string constant.

ASCII

In our definition of string data, string characters were described as the set of characters which can be represented with ASCII codes. The PC*jr* cannot store characters; it can only store numbers. Before characters can be stored, they must be converted to numbers. Computers use special numeric codes to store characters. Most personal computers including the PC*jr* use a coding system known as ASCII (American Standard Code for Information Interchange).

The PC*jr* uses codes which differ slightly from the standard ASCII code set. The codes used by the PC*jr* are listed in appendix A. Notice that ASCII code 77 is used to represent the character M. A different ASCII code, 109, is used to represent the lower case m.

Numeric Data

Numeric data can be defined as information represented with numbers that can be used in performing calculations. Numeric data is stored and operated upon in a different manner than string data.

A numeric constant can be defined as a numeric data item with a fixed value. In the remainder of this book, we will refer to a string constant as a string and a numeric constant as a number. The following are examples of numeric constants:

-7487	14.72#	36.9
29!	1.4E7	37481.58342

Numeric constants cannot include commas. For example, 10,000 would be an illegal numeric constant.

BASIC classifies numeric constants as **integers, fixed-point** numbers, **floating-point** numbers, **hexadecimal** numbers, and **octal** numbers. Each of these numeric data types will be discussed in the following sections.

Integers

An **integer** is a number without a decimal portion. Integers can either be positive or negative. The following are examples of integers:

-1134	0	1
-1	17945	+32

Integers can range from -32768 to +32767. Negative integers are preceded with the (-) sign. Positive integers can be preceded with the (+) sign, although integers without a sign are assumed to be positive.

Fixed-Point Numbers

Fixed-point numbers can be defined as the set of positive and negative real numbers. Fixed-point numbers contain a decimal portion. The following are examples of fixed-point numbers:

79.8394	-1.01
7.0	14376.91782

Floating-Point Numbers

Floating-point numbers are represented in exponential notation. A number in exponential notation takes the following format:

$$\pm x \text{ E} \pm yy$$

$\pm$ is an optional plus or minus sign.

x can either be an integer or fixed-point number. This portion of the number is known as the coefficient or mantissa.

E stands for exponent. D can also be used instead of E to specify a double-precision* floating-point constant. Either D or E can be interpreted as "times ten to the power of the exponent".

yy is a one or two digit exponent. The exponent gives the number of places that the decimal point must be moved to give its true location. The decimal point must be moved to the right with the positive exponents. The decimal point is moved to the left with negative exponents.

The following are examples of floating-point numbers and their equivalent notation in fixed-point:

Floating-Point	**Fixed-Point**
17E-4	.0017
237.9823E-9	.0000002379823
173.1E5	17310000.0

Any number in the range of 10E-38 to 10E+38 can be represented in floating-point form.

* Double-precision is explained in a later section.

Hexadecimal Numbers

In mathematics, base 10 or decimal notation is normally used. All of the previous examples were in base 10.

Hexadecimal numbers use 16 as a base rather than 10 as in decimal notation. The digits 10, 11, 12, 13, 14, and 15 are represented with the letters A, B, C, D, E, and F, respectively. Hexadecimal numbers are prefixed with &H in BASIC. The following are examples of hexadecimal numbers and their equivalent decimal values:

&HA4	164
&H231	561
&HA1A	2586

Octal Numbers

Octal numbers use 8 as the base. The digits 0 through 7 are used in octal. Octal numbers are prefixed with &O in BASIC. The following are examples of octal numbers and their equivalent decimal values:

&O457	303
&O12	10
&O20	16

Numeric Precision

Precision in the context of numeric data can be defined as the number of significant digits used in representing the data. In BASIC, numeric data may be stored as integers, single-precision numbers, or as double-precision numbers. Each of these are stored differently in the PC*jr*'s memory, so the distinction is important.

As we discussed earlier, integers are whole numbers in the range -32768 to 32767. A **single-precision** value can be defined as a non-

integer numeric value with a maximum of seven digits. A **double-precision** value can be defined as any value with 8 or more digits. A maximum of 16 digits will be printed for a double-precision constant. Two bytes are required to store an integer value. Four bytes are required for a single-precision value. Eight bytes are required for a double-precision value.

Any of the following would be evaluated as single-precision constants:

1.78
147.986
94387
1.01E6
7!
1978.24871!

From our definition of a single-precision value, it is evident that the first two numeric examples, 1.78 and 147.986, are single-precision values. Since they contain a fractional portion, they are non-integers. Since both contain less than seven digits, they fit our definition for a single-precision value.

On the surface, the third example, 94387, appears to be an integer as it does not contain a decimal portion. However, since this value lies outside of the allowed range for integers (–32768 to 32767), it is regarded as single-precision.

The fourth example is written in floating-point form with E used to indicate exponentiation. By definition, any values represented in exponential form using E are regarded as being of single-precision.

The trailing exclamation point (!) is used to force a value into single-precision that would otherwise be regarded as an integer or double-precision value. This is shown in our last two examples. Although 7 would normally be an integer, the inclusion of the trailing exclamation point forces it into single-precision. Likewise, 1978.24871, which would otherwise be a double-precision value, is forced into single-precision by including the exclamation point as a suffix.

The following are examples of double-precision values:

4.98372443217
37854.98321
3.248D-06
7#
14.738#

From our definition, we can see that the first two values are double-precision, as they each contain over seven digits. The third example is written in floating-point form using D to indicate exponentiation. By definition, any values represented in exponential form using a D are regarded as being of double-precision.

The trailing number sign (#) is used to force a value into double-precision that would otherwise be regarded as an integer or single-precision value. This is shown in our last two examples.

Variables

So far in this lesson, we have discussed BASIC's different types of data -- string and numeric.

However, we have only discussed representing data as a constant. The value of a string or numeric constant such as "JIM HILL" or 27.92 remains the same.

Data can also be represented by using a **variable.** A variable can be defined as an area of memory that is represented with a name. That name is known as the **variable name.** The information stored in the memory area defined by a variable name can vary (hence the name variable) as BASIC commands or statements are executed. The data currently stored in the memory area defined by a variable is known as the variable's **value.**

Variable Names

BASIC allows variable names of up to 40 characters in length. A variable name must begin with a letter of the alphabet (A-Z) followed by additional letters, digits, or decimal points. Blank spaces are not allowed within a variable name. The following are examples of valid BASIC variable names:

PA4.1	A
X123	TOTAL.JUNE
QR37A	Z17

A variable name may not duplicate a BASIC reserved word (see appendix B). However, a variable name may incorporate a reserved word as part of its name.* Therefore, although the following would be invalid variable names:

NEW DATA PRINT

the following variable names would be valid:

NEW.PHONE DATA.X PRINTNAME

Variables, like constants, can either be numeric or string. Numeric variables can be integer, single-precision, or double-precision.

* The exception to this rule is FN. A variable name may not begin with FN.

A variable type can be declared by using a type identification character. The type identification characters are as follows:

% = integer
! = single-precision
= double-precision
$ = string

For example, the following variable names would be declared as string, single-precision, and integer, respectively:

ANCIENT$ B12! JACK%

If a variable type character is not specified, the variable type is assumed to be single-precision.

Assigning Values to Variables with the LET Statement

Now that we have defined what a variable is and how variable names are assigned, we are ready to learn how to assign a value to a variable. BASIC's LET command is used to assign a value to a variable. LET statements are also known as **assignment** statements. An example of a LET statement is given below:

```
LET A$ = "John"
```

The LET command causes the value in the right-hand side of the equation to be assigned to the variable on the left-hand side.

The LET command is unique, as the reserved word LET need not actually be included in the LET statement. This is evidenced in the following example:

```
LET A$ = "John"
PRINT A$
A$ = "Sam"
PRINT A$
```

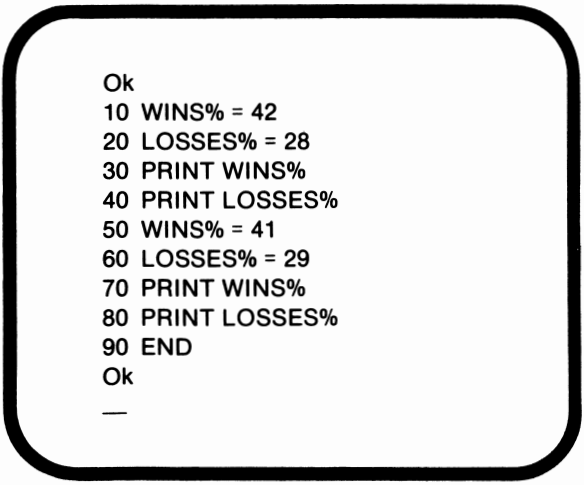
By entering the preceding statements, it is evident that if the LET command is not included in the assignment statement, the variable is still assigned a new value.

The value assigned to a variable can either be a constant or another variable. This is shown in the following example:

```
Ok
10 A$ = "John"
20 B$ = A$
30 PRINT A$
40 PRINT B$
50 END
RUN
John
John
Ok
_
```

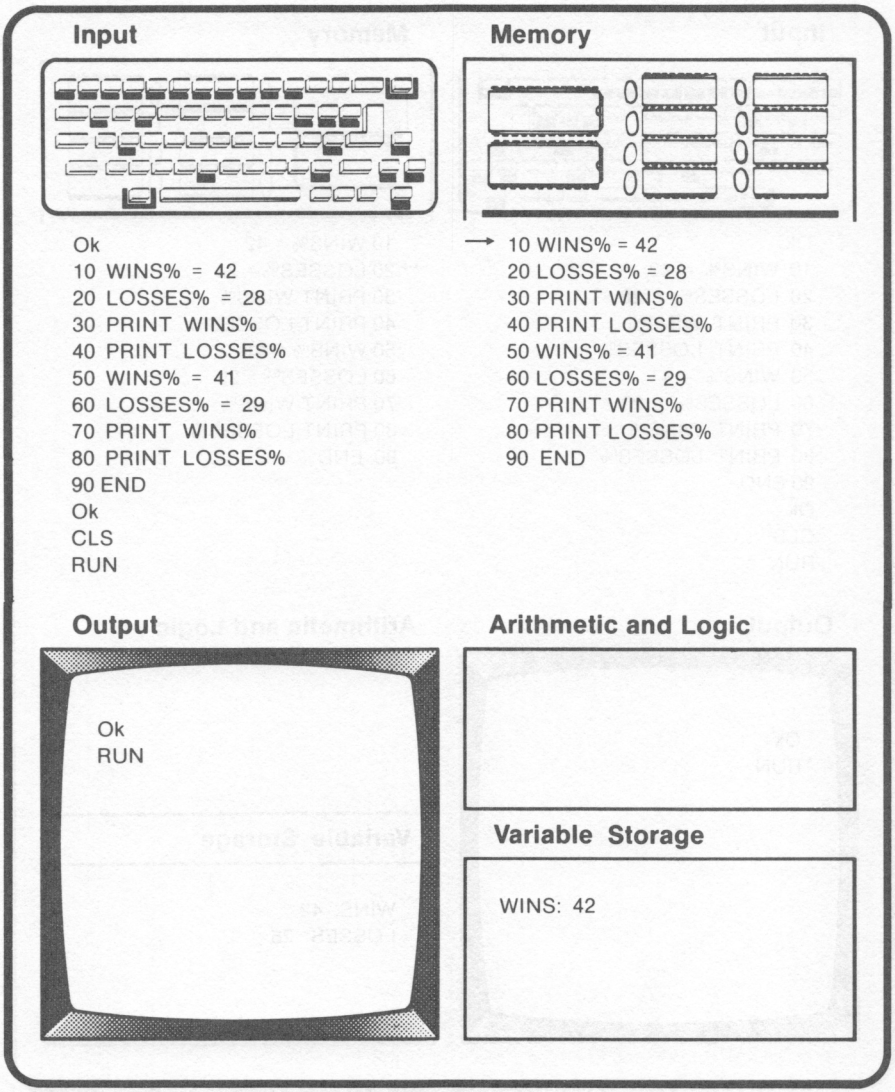
How Variables are Processed

Now that we have gained more understanding of what variables are and how they are assigned values, let's examine how the PC*jr* processes and stores variables and their values. Suppose that you entered the following program:

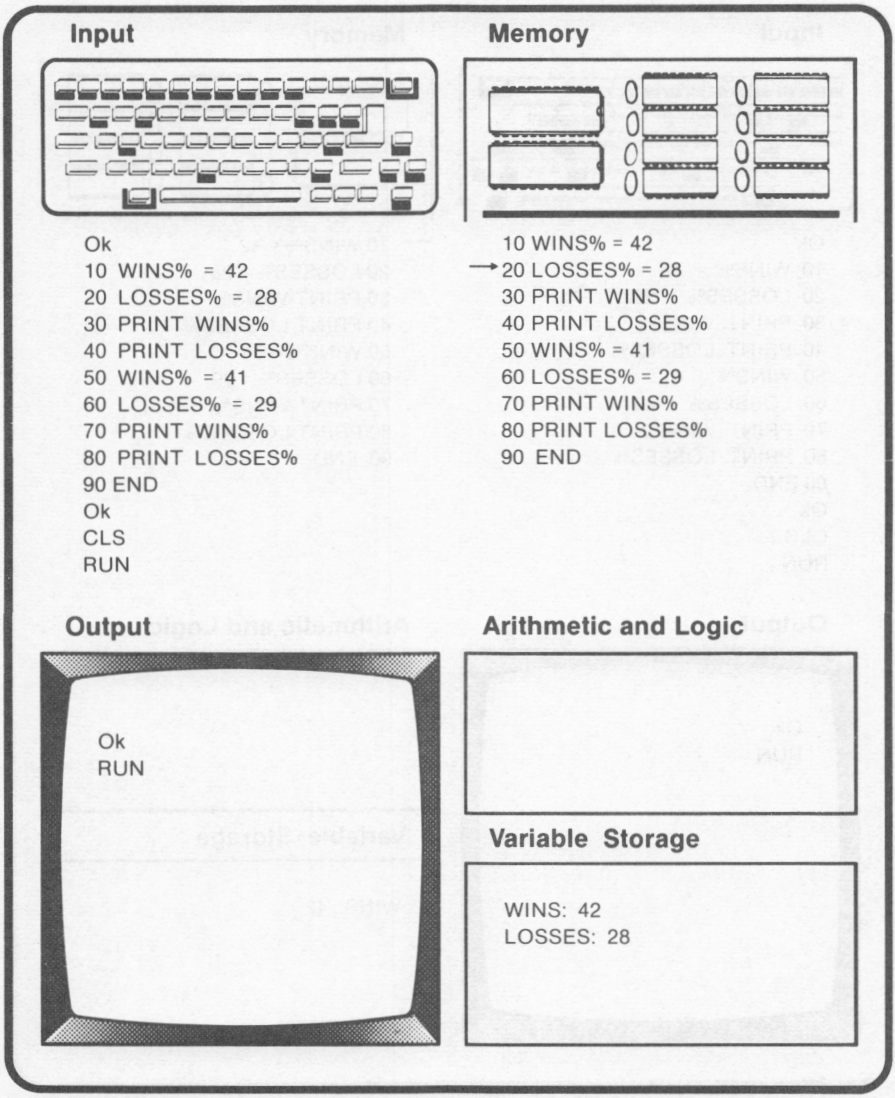


When this program is run, it will be executed as a series of eight steps. These steps will be depicted on the following pages.

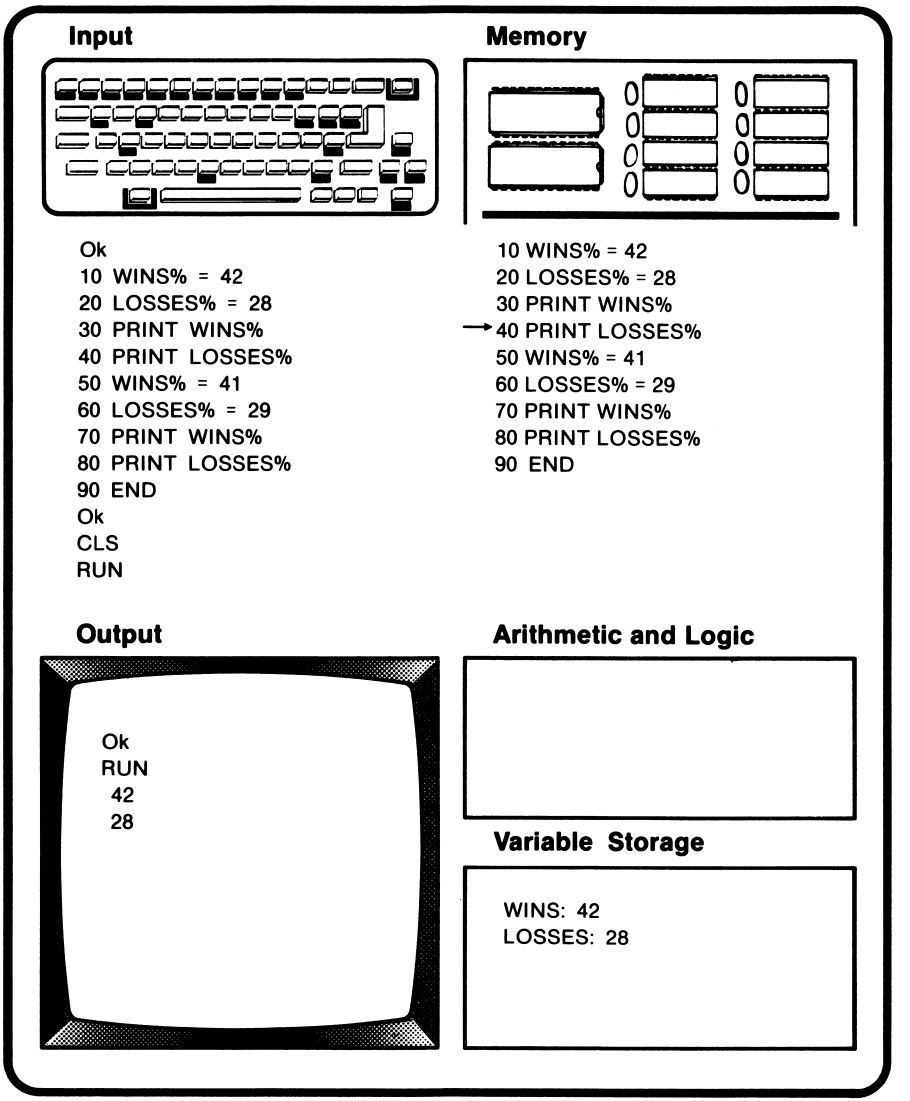
When the program is run, line 10 will be executed first. Line 10 is an assignment statement in which the integer variable named WINS% is assigned the integer value 42. Notice from the illustration that an area in variable storage memory is assigned the name WINS%, and the value assigned to WINS% is stored in that area.



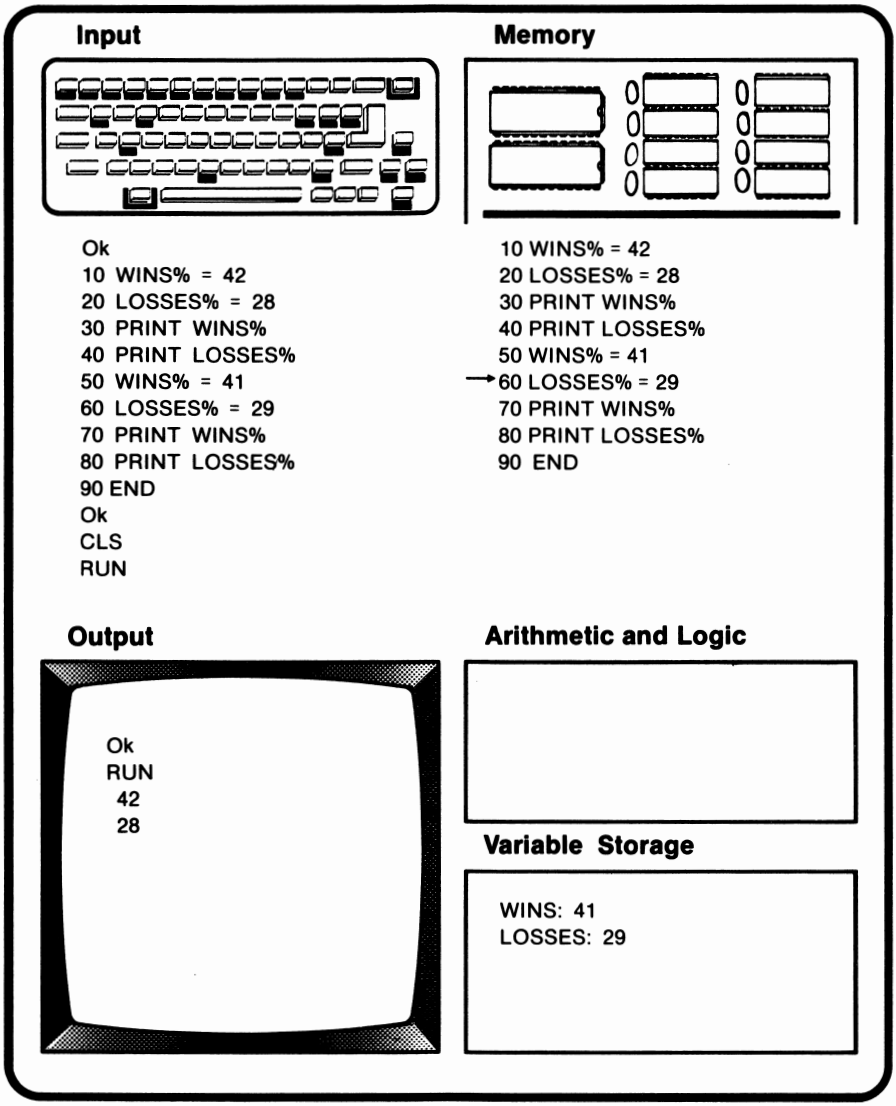
When line 20 is executed, LOSSES% is assigned an area in variable storage, and the value indicated in the assignment statement, 28, is stored there.



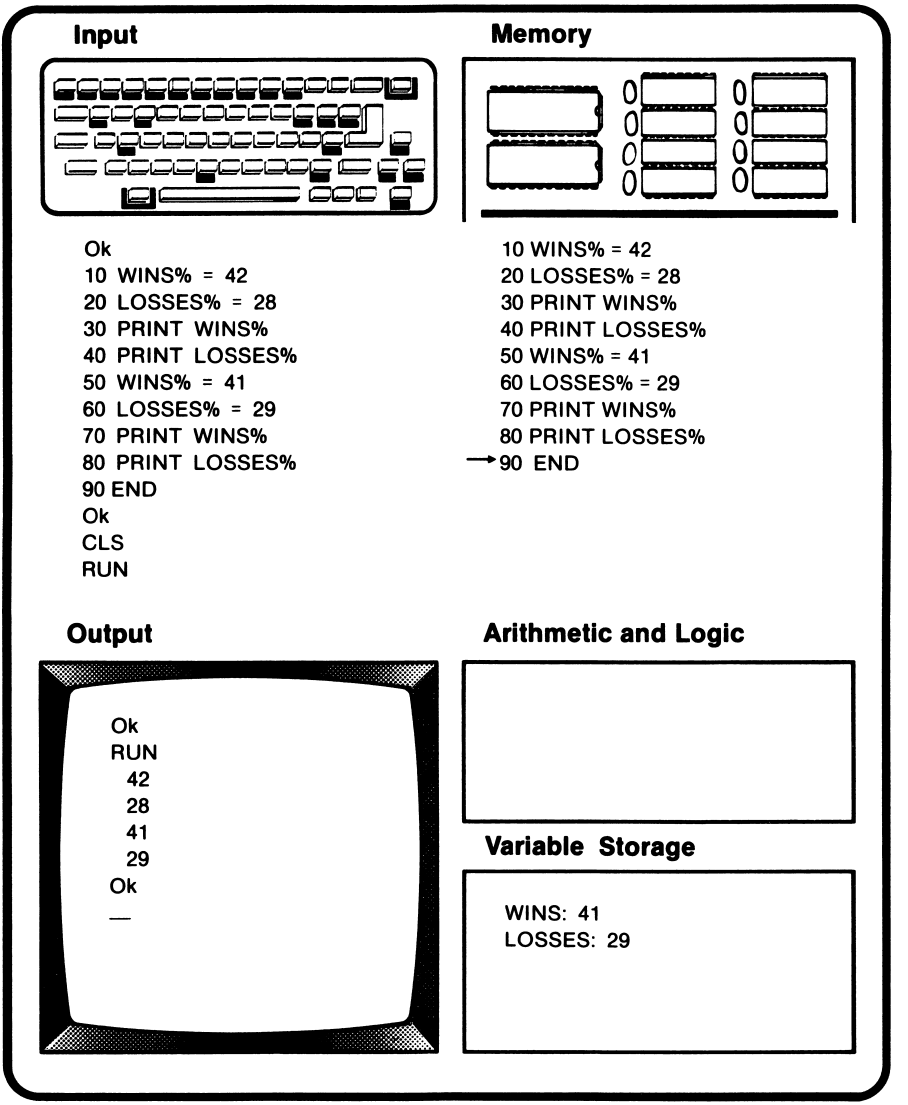
In lines 30 and 40, the values stored in memory for WINS% and LOSSES% are displayed on the screen.



In line 50, the variable WINS% is assigned a new value. In line 60, the variable LOSSES% is also assigned a new value. Notice from the illustration below that the new values replace the previous values in variable storage.



The new values for WINS% and LOSSES% are displayed when lines 70 and 80 are executed.



One final point to keep in mind is that execution of the NEW command not only erases a program from memory, but it also clears all variable values as well.

The main point to be gained from this section is that a variable name references an area in memory, and that the values stored in that area can vary.



True or False

1. The integer numeric value 9 is represented internally with the ASCII code 57.
2. The PC*jr* allows quotation marks to be included within string constants.
3. Hexadecimal numbers use 16 as their base.
4. Variable values must remain constant in memory.
5. The numeric value 7# will be regarded as being of double-precision.

Multiple Choice

1. ASCII codes are used to represent:
   - A. Numeric values
   - B. Single-precision numbers
   - C. Octal values
   - D. String characters
   - E. None of the above

2. Which of the following is a fixed-point number?

   A. 27
   B. "14.1"
   C. 1.7E+7
   D. 137.931248732!
   E. None of the above

3. Which of the following is a floating-point number?

   A. &HAF
   B. 1.7D7
   C. &O20
   D. 1.7936#
   E. None of the above

4. Which of the following variable names are valid?

   A. 7D%
   B. FNX$
   C. CLS!
   D. A37X9!
   E. All of the above

5. Which of the following statements would assign the string constant "Phil" to a string variable?

   A. LET NAME = PHIL
   B. A$ = PHIL
   C. X$ = PHIL$
   D. All of the above
   E. None of the above

Essay

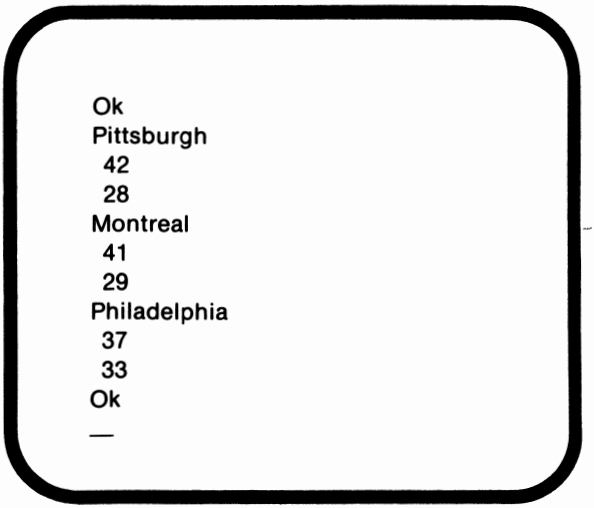
1. Define string data and numeric data.
2. Define the term variable.
3. Define the term numeric precision.

Computer Exercise

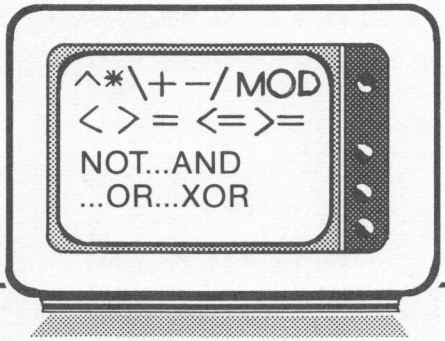
1. Write a program to assign the following constants to variables:

"Pittsburgh"	"Montreal"	"Philadelphia"
42	41	37
28	29	33

and display this information in the following format:



Note: A blank space is automatically output prior to a numeric value when that value is specified with a PRINT statement.



Operators

lesson 12

Lesson Goals

- *Learn what expressions are and how to use them*
- *Learn how to use BASIC's arithmetic operators*
- *Learn how to use BASIC's relational operators*
- *Learn how to use BASIC's logical operators*
- *Learn in what order the PCjr evaluates expressions*

Introduction

In this lesson, we will work with operators and expressions. An operator is a sign or phrase which represents an action that the computer is to perform. An operator is generally used as part of an **expression.** An expression is used to combine values, also known as operands, to produce a new value. Operators usually have two operands, one to the left of the operator and one to the right. The exception is negation, which we will discuss later in this lesson.

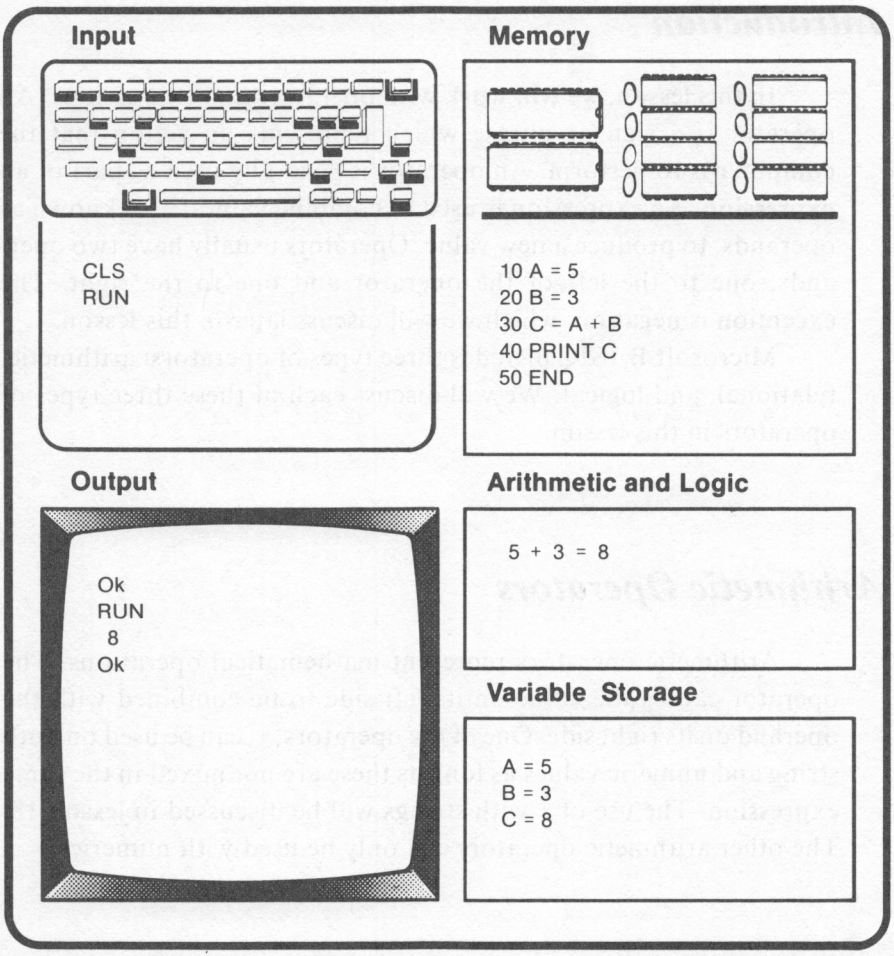
Microsoft BASIC includes three types of operators: arithmetic, relational, and logical. We will discuss each of these three types of operators in this lesson.

Arithmetic Operators

Arithmetic operators represent mathematical operations. The operator causes the value on its left side to be combined with the operand on its right side. One of the operators, +, can be used on both string and numeric values as long as these are not mixed in the same expression. The use of + with strings will be discussed in lesson 18. The other arithmetic operators can only be used with numerics.

Addition (+)

The plus sign (+) is used to add numerics. This is shown in the following example:



In this program, 5 was assigned to A and 3 to B. When line 30 was executed, the values stored in the variables A and B were added together, resulting in 8. This value was then assigned to C. Line 40 caused the value in C to be printed on the screen.

Exponentiation (^)

The caret (^) is used in BASIC to indicate exponentiation. In exponentiation, one number, the **base,** is raised to a new value. The new value is obtained by multiplying the base times itself. The number of times the base is to be multiplied by itself is indicated by the **exponent.**

For example, in the following expression:

```
X ^ 2
```

the numeric variable X would be evaluated as X multiplied by X. X is the base, and 2 is the exponent.

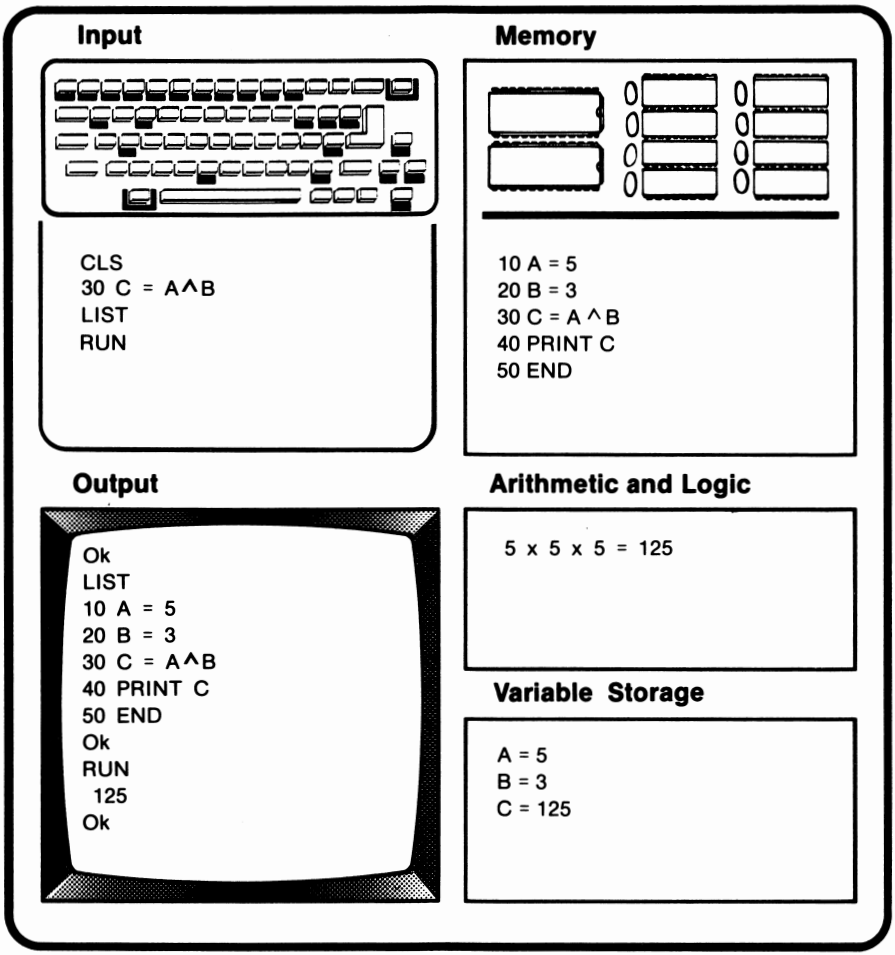
Notice that exponentiation is consistent with our definition of an expression. An operator, ^ , is used to combine the left operand, the base, with the right operand, the exponent, to produce a new value.

You might be more familiar with exponentiation as represented in its standard algebraic format as shown below:

$$X^2$$

Again, X is the base, and 2 is the exponent.

The use of ^ in a BASIC program can be seen by editing and running the previous program as shown:

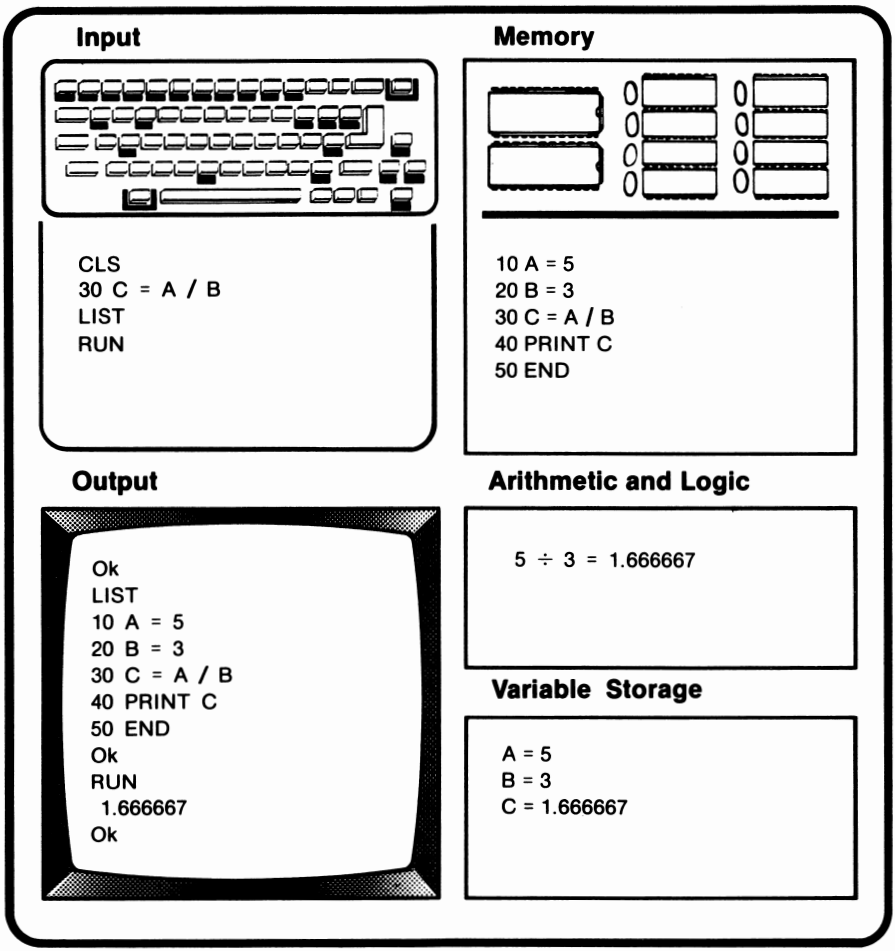


Notice that when line 30 was executed, the 5 was cubed. In other words, 5 was multiplied by itself three times.

Floating Point Division (/)

Floating point division acts as does ÷ in mathematics. The first

operand is simply divided by the second. This can be seen in the following illustration:

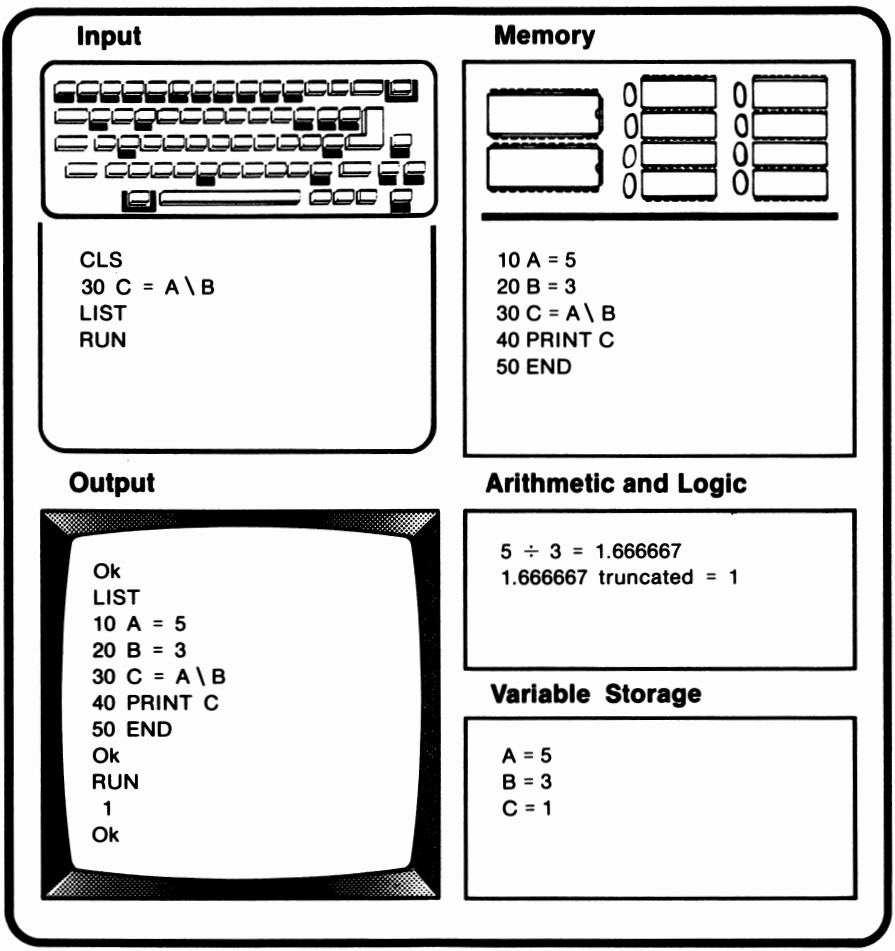


Floating point division is an appropriate name, because the decimal point can move, or float, as needed. If no decimal point is necessary, none will be printed.

Integer Division (\ )

Integer division results in two integer operands being divided with another integer as the result. During execution, the operands, if not already integers, are rounded to integers. The division is then performed, and the quotient is truncated to an integer. In other words, any numbers after the decimal point are just thrown away.

Integer division is like asking, "If I have five hamburgers and there are three people at my picnic, how many hamburgers are there per person?" The answer is one. Integer division will give the same result. This can be seen in the following example:

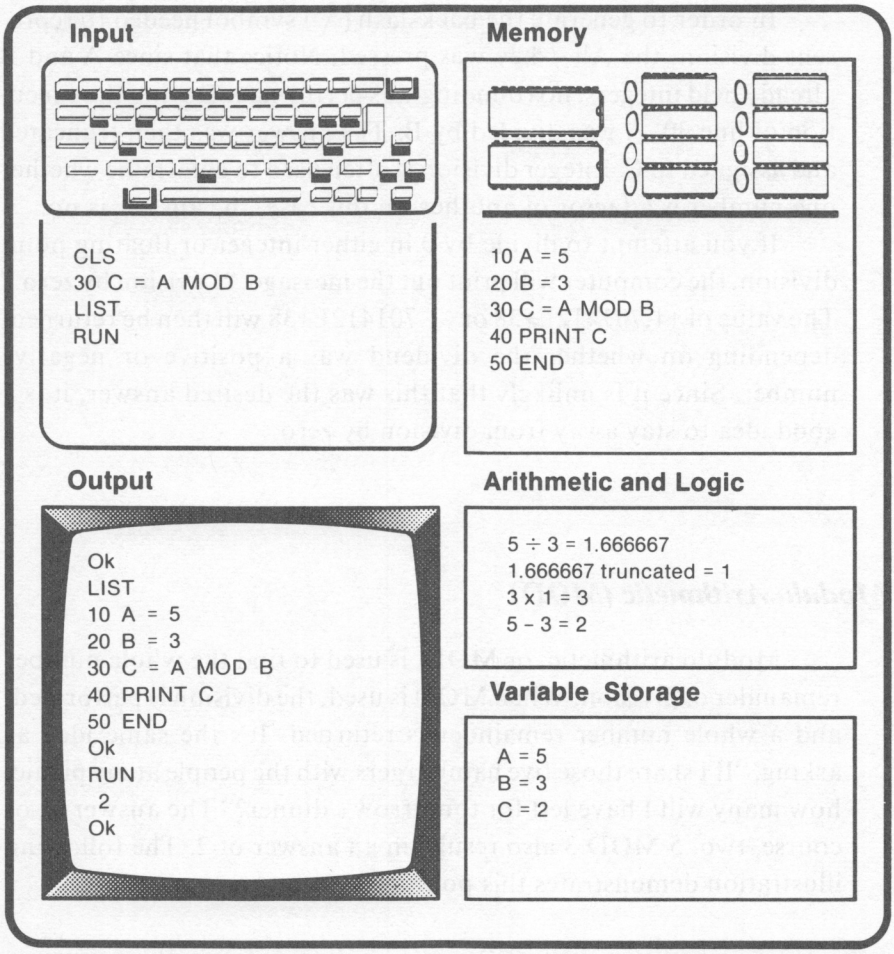


In order to generate the backslash (\ ) symbol needed to represent division, the Alt-/ key was pressed. Notice that since A and B already held integers, no rounding was performed. During the execution of line 30, A was divided by B. The answer was then truncated and assigned to C. Integer division is often used to determine whether one number is a factor of another. In this case, the answer is no.

If you attempt to divide by 0 in either integer or floating point division, the computer will print out the message, "Division by zero." The value of +1.701412E+38 or -1.701412E+38 will then be returned, depending on whether the dividend was a positive or negative number. Since it is unlikely that this was the desired answer, it is a good idea to stay away from division by zero.

Modulo Arithmetic (MOD)

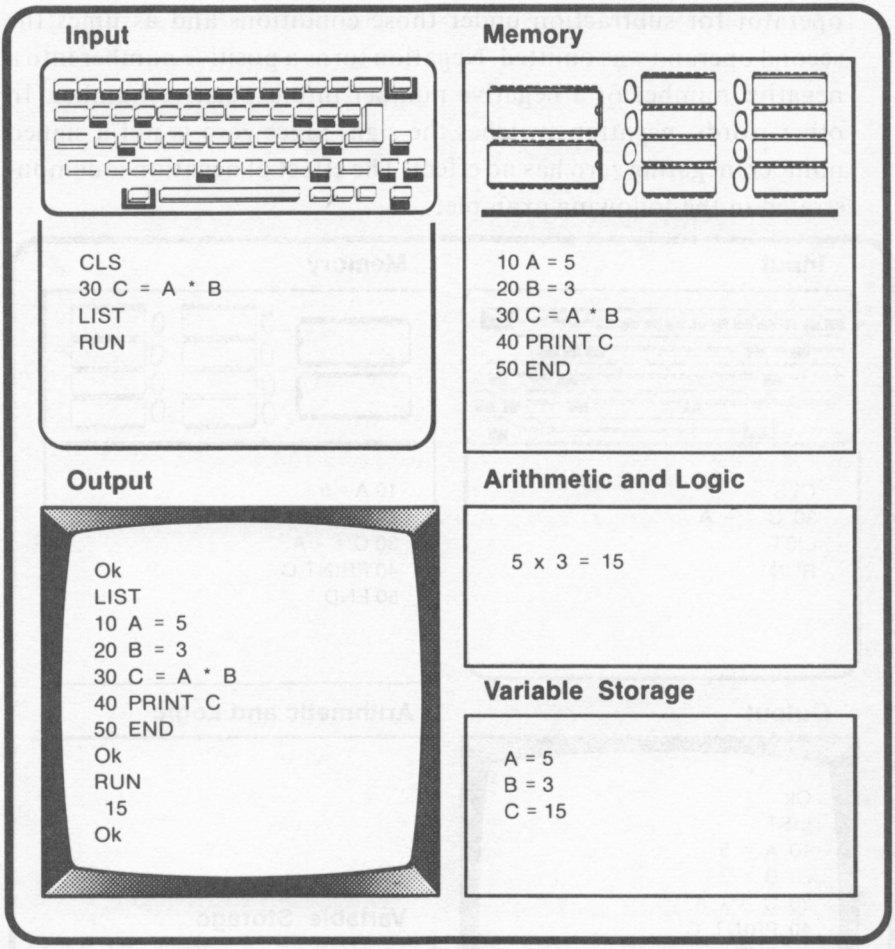
Modulo arithmetic, or MOD, is used to find the whole number remainder of division. When MOD is used, the division is performed, and a whole number remainder is returned. It's the same idea as asking, "If I share those five hamburgers with the people at my picnic, how many will I have left for tomorrow's dinner?" The answer is, of course, two. 5 MOD 3 also results in an answer of 2. The following illustration demonstrates this point:



The operands in a MOD expression do not have to be integers. MOD causes noninteger operands to be rounded. Blank spaces must separate MOD from its operands. Spaces are not mandatory with the other arithmetic operators. It is, however, a good programming practice to include a blank space on either side of an arithmetic operator.

Multiplication (*)

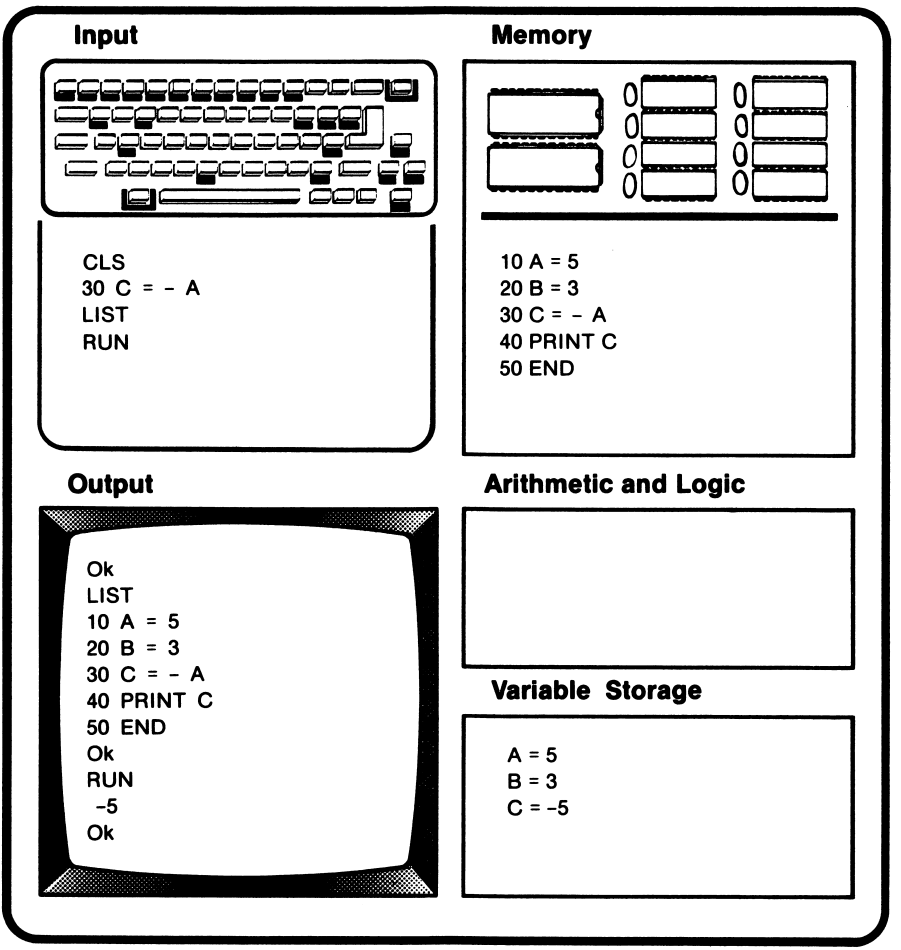
The symbol for multiplication is the asterisk (*). This operator is illustrated in the following example:



Multiplication can be substituted for exponentiation. Substituting multiplication, however, can be rather inefficient, especially when a number is to be raised to a large power.

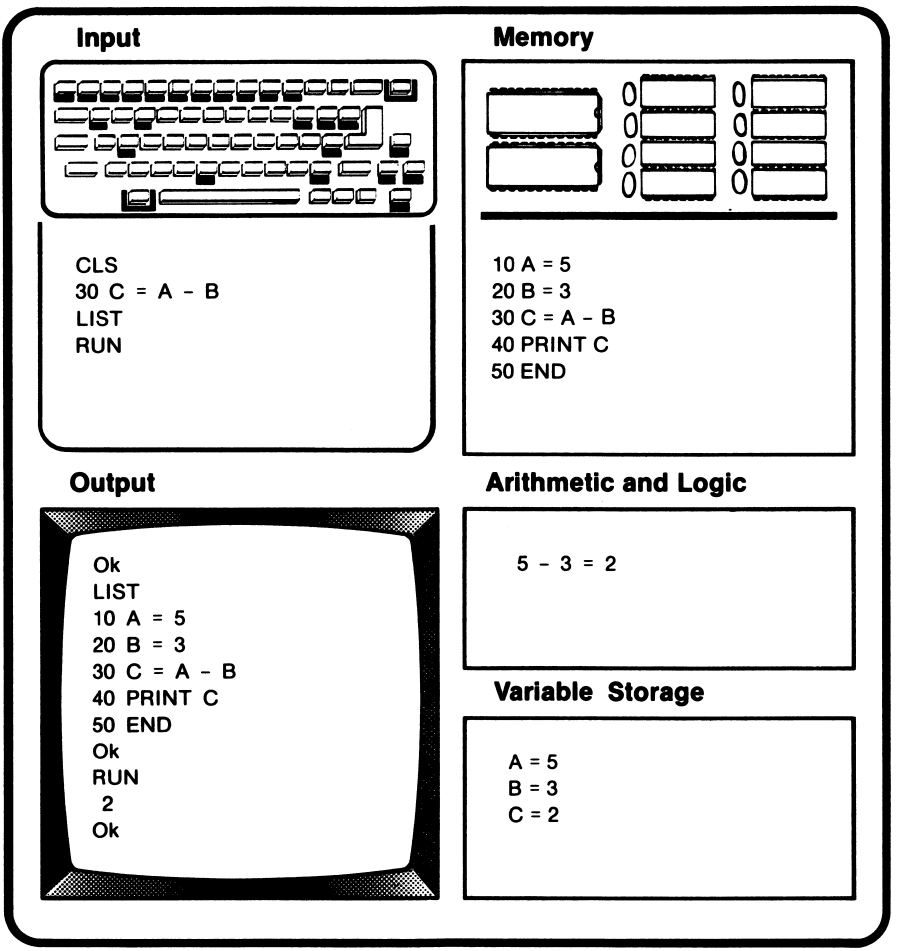
Negation (-)

Negation and subtraction both use the minus sign (-). However, negation includes only one operand, which is placed to the right of the operator. Placing the operand to the left of the operator will result in a missing operand error, since the computer interprets the - as the operator for subtraction under those conditions and assumes the second operand was omitted. Negation turns a positive number into a negative number or a negative number into a positive number. In other words, negation switches the sign. Since zero is not a signed number, negating zero has no effect. The effect of negation is demonstrated in the following example:



Subtraction (-)

Subtraction in BASIC is identical to subtraction in mathematics. This is illustrated by the following example:



Order of Evaluation

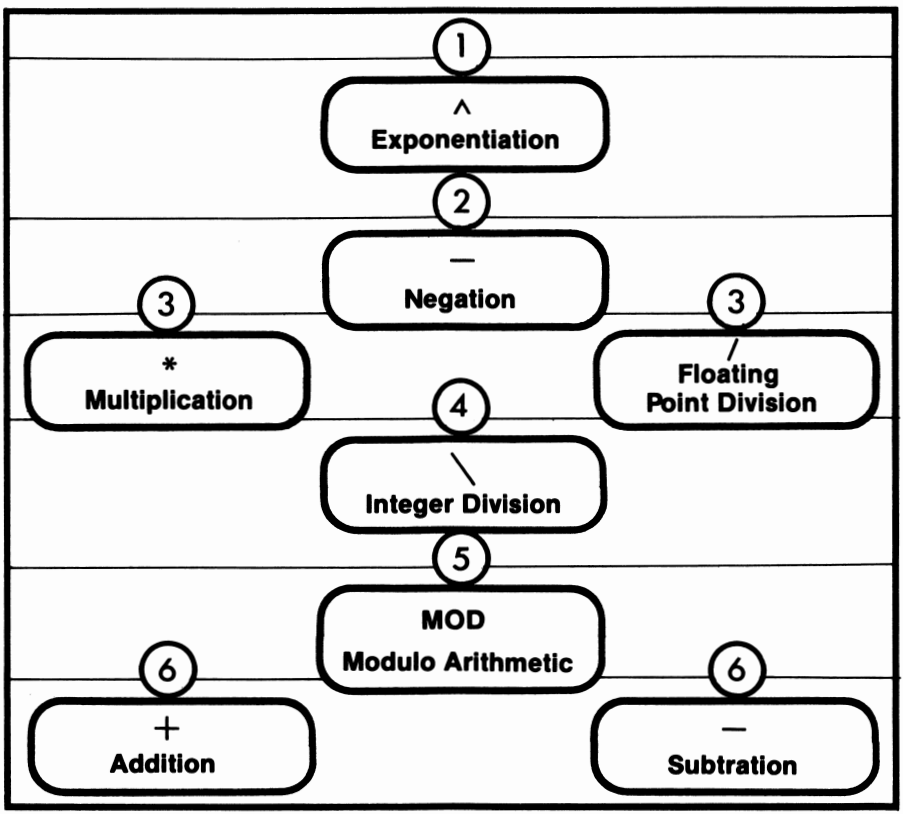
In the previous examples, operators were used to form **simple expressions.** A simple expression is one which contains one operator and one or two operands. Operators can also be used to form **com-**

pound expressions. A compound expression consists of two or more simple expressions. 5 + 5 – 3 is an example of a compound expression. When compound expressions are used, BASIC follows a set of rules which specify which operation is to be performed first, second, etc. These rules are known as the **order of operations.**

A general rule in the order of operations is that operations are performed from left to right. In other words, if we tell the computer to add three numbers, for example 5 + 7 + 2, five and seven are added first, and that value and two are added.

Certain operators have a higher priority than others. This alters the general rule. Table 12.1 lists the operator priorities from the highest level to the lowest level.

Table 12.1. Order of evaluation of arithmetic operators



Operators with a higher priority level are evaluated before operators with a lower priority. Within each level of priority, expressions on the left are evaluated before those on the right.

Mixing Variable Types in Arithmetic Expressions

Generally, it is best not to mix variable types. Sticking to the same types of variable within an expression cuts down on running time, saves space in memory, and decreases the probability of programming errors.

If variable types are mixed, the value will, if possible, be changed during execution to fit the specified variable type. For example, if 5.7 is assigned to an integer variable, five will be rounded to six and then stored. If the value cannot be changed, an error message will appear on the screen. One possible error is attempting to add a string, "Hello", to a numeric, 5.

Relational Operators

BASIC includes six relational operators. Relational operators are used to compare two values, both of which are either numeric or string. A 0 is returned if the comparison is false, and a –1 is returned if the comparison evaluates to true. The relational operators are listed in table 12.2.

Whenever a relational operator is used, it should be separated from its operands by one space on either side. Again, the spaces are not mandatory, but it is good programming style to include them.*

* Notice that the IBM PC*jr* does not accept LE, LT, GE, GT, EQ, and NE as operators.

Table 12.2. Relational Operators

Relational Operators	
<	less than
<= or =<	less than or equal to
>	greater than
>= or =>	greater than or equal to
=	equal to
>< or <>	not equal to

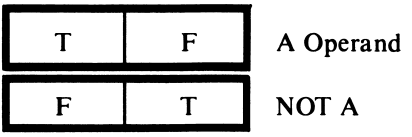
Strings cannot be compared with numerics, but strings can be compared with other strings. If the two strings do not have the same number of characters, the shorter string is considered to be the lesser. For example, "B" is evaluated as less than "AA". If the strings are the same length, the ASCII codes for the characters in the string are compared. The string which has the lower code number in the earliest position is evaluated as the lesser. For example, "ABCD" is considered to be less than "ACBD". Blank spaces do count; each has an ASCII code of 32. "AAA" and "aAA" are not equal, because capital and small letters have different ASCII codes. Capital letters have lower code values than their small letter counterparts. A list of ASCII characters can be found in appendix A.

Logical Operators

BASIC includes four commonly used logical operators. They are most often used to compare the results of relational operators. Logical operators are also known as Boolean operators. A value of -1 is returned if the comparison is true, and a value of 0 is returned if the comparison is evaluated as false. Be careful to use only -1 and 0 as operands. Using other operands can cause inaccurate results.

NOT

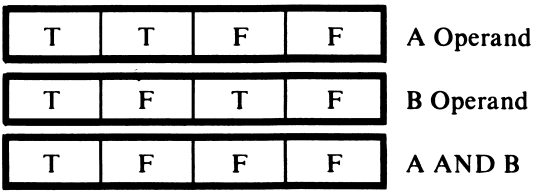
NOT acts as the logical complement. In other words, NOT changes a true value to false or a false value to true. This is illustrated by the following logic diagram:



NOT is most often used when something is to occur if the condition is false. An example would be only considering a person for a basketball team if that person was not shorter 5'8", or eating a hamburger for dinner only if you didn't eat one for lunch.

AND

When AND executes, if both operands are true, a value of true is returned. Any other combination results in a false value being returned. This can be seen in the following logic diagram:



AND only evaluates to true if both operands are true. An example of AND reasoning is Chris and Jim who will only purchase an item at the grocery store if both people want to buy it. If only one of them wants the product, they won't buy it.

OR

When OR executes, a value of true will be returned if either of the operands is true. The following diagram illustrates the operation of OR:

T	T	F	F	A Operand
T	F	T	F	B Operand
T	T	T	F	A OR B

OR is most often used when something is to be done as long as any one of the conditions is true. An example is Chris and Jim who decide they will buy a book as long as one of them wants to read it. If they both want to read the book, they will still buy it.

XOR

XOR is the exclusive OR operator. XOR only returns a true value if just one of the operands is true.

T	T	F	F	A Operand
T	F	T	F	B Operand
F	T	T	F	A XOR B

Notice that XOR returns a false value if both operands are true. Jesse, who wants either chicken or fish for dinner, but not both, is using XOR reasoning.

Order of Evaluation

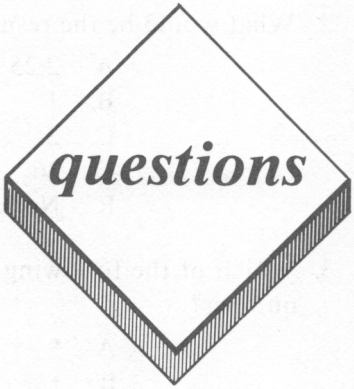
Earlier in this lesson, we examined the order of evaluation for arithmetic operators. Additional rules are applied when the compound expression contains logical or relational operators. These expressions are also evaluated from left to right within priority levels. Arithmetic expressions are evaluated first, followed by relational expressions. Logical expressions are evaluated last. The order of evaluation among BASIC's arithmetic, relational, and logical operators is summarized in table 12.3.

Table 12.3. Order of evaluation

	Order of Evaluation		
Arithmetic:	^	Exponentiation	
	—	Negation	
	* /	Multiplication	Floating point division
	\	Integer division	
	MOD	Modulo arithmetic	
	+ —	Addition	Subtraction
Relational:	=	Equal	
	<> ><	Not equal	
	<	Less than	
	>	Greater than	
	<= =<	Less than or equal to	
	>= =>	Greater than or equal to	
Logical:	NOT		
	AND		
	OR		
	XOR		

The order of evaluation can be changed by enclosing an expression within parentheses. The expression within the parentheses will be evaluated first. For example, when the expression, 5 / (4 + 6) is evaluated, four and six are added. Then five is divided by ten, the sum of four and six. The answer is 0.5. In contrast, when the expression, 5 / 4 + 6 is evaluated, five is divided by four, resulting in 1.25. Then 1.25 and 6 are added. The answer obtained is 7.25.

If parentheses are placed where they aren't necessary, the parentheses will be ignored during execution. In the previous example, if (5 / 4) + 6 had been the expression, instead of 5 / 4 + 6, the order of evaluation is not altered by the parentheses, and the computer operates as if the parentheses were not there. It is good programming style to use parentheses whenever the parentheses will make the order of evaluation clearer to another person, regardless of whether or not the computer needs them. Whenever you use parentheses, make sure you use the same number of left parens and right parens. Otherwise, the run will stop due to a syntax error.



True or False

1. Both arithmetic operators and relational operators use mathematics to combine operands.
2. MOD returns the whole number remainder of division.
3. Relational operators return a value of 0 if false and 1 if true.
4. AND returns a value of true only if both operands are true.
5. Whatever expression is included within parentheses will be evaluated first.

Multiple Choice

1. In the compound expression, 5 < 23 AND 6 + 9 >= 25, which of the following simple expressions would be evaluated first?

   A. 5 < 23
   B. 23 AND 6
   C. 6 + 9
   D. 9 >= 25
   E. None of the above

2. What would be the result of the expression, 9/4?

   A. 2.25
   B. 1
   C. 2
   D. 36
   E None of the above

3. Which of the following symbols will raise a number to another number?

   A. *
   B. +
   C. MOD
   D. ^
   E. None of the above

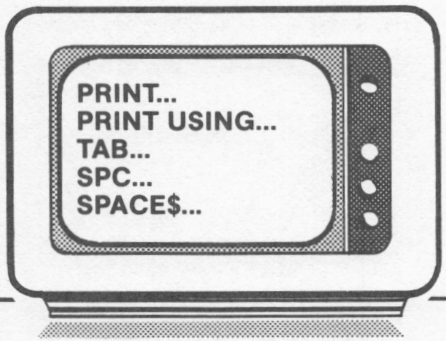
4. Which of the following logical operators will be evaluated as true if the A operand is true and the B operand is true?

   A. AND
   B. XOR
   C. NOT
   D. None of the above
   E. All of the above

Computer Exercises

1. Using the program you wrote for Computer Exercise 1 on page 128, compute and display the average grade.
2. Use the computer to evaluate each of the following expressions:

   a. 6 + 9 / 3
   b. (6 + 9) / 3
   c. 5 < 23 AND 6 + 9 >= 25
   d. 5 < 23 OR 6 + 9 >= 25



Outputting Data

lesson 13

Lesson Goals

- *Learn about the uses of PRINT to handle more than one piece of data at a time*
- *Learn the uses of the PRINT USING command*
- *Learn how to use formatting characters with PRINT USING*
- *Learn how to use the formatting functions of TAB, SPC, and SPACE$*

Introduction

In lesson 6, we discussed how to use PRINT to output one item at a time. In this lesson, we will discuss how to output more than one item at a time.

We will also look at how to format output so that it is appealing to the eye. In lesson 6, this goal was accomplished by entering the needed number of spaces into the string. This method isn't very efficient. The formatting of output can also be handled by the PRINT USING statement, formatting characters, and formatting functions. We will discuss each of these options in this lesson.

PRINT

Let's look at what happens when we try to ouput three items with a single PRINT command. Notice the following illustration:



There are no spaces in the line of output. It appears that the computer treated the individual items as one string. Let's examine what happens when a space is placed between each entry:

```
Ok
PRINT "Bacon" "Lettuce" "Tomato"
BaconLettuceTomato
Ok
```

Again, the items appear as one string. If the spaces are replaced by commas, the situation changes:

```
Ok
PRINT "Bacon","Lettuce","Tomato"
Bacon                    Lettuce
Tomato
Ok
```

The insertion of commas between the items caused the output to appear with one item per print zone. A print zone consists of 14 spaces. The comma, when used with PRINT, spaces the strings by print zones. Normally, the screen has 40 columns, but the enhanced version of the PC*jr* can be set to 80 columns by entering the command WIDTH 80. The screen can be returned to 40 columns by entering WIDTH 40. A 40 column screen has two print zones per line. With the screen set to 80 columns, there are four print zones per line. We will look at this is in more depth in lesson 20.

We can alter the placement of the strings by changing the commas to semicolons. The following example demonstrates this point:

```
Ok
PRINT "Bacon";"Lettuce";"Tomato"
BaconLettuceTomato
Ok
```

It appears that using semicolons causes the three strings to be treated as though they were one. In this instance, however, the computer considers the items to be three separate strings. This point can be more clearly seen by substituting numbers for the strings, as in the following example:

```
Ok
PRINT 10 20 30
 102030
Ok
PRINT 10;20;30
 10  20  30
Ok
```

Numbers are printed with a blank space automatically inserted after each number. A blank space is also inserted prior to each number if the number is zero or positive. If the number is negative, the blank space is replaced by a negative sign.

With strings, blank spaces are not automatically inserted when a semicolon is used to divide the items. If you want spaces separating the strings, the desired number of spaces can be included within each string, as in lesson 6. We will discuss other methods later in this lesson.

A final point to note is that after each PRINT command, the cursor advances to the leftmost position on the next output line. This

is known as the carriage return/line feed or CR/LF. If you want the output of separate PRINT commands to appear on the same line, the CR/LF can be suppressed by ending the statements with a comma or a semicolon. This can be seen in the following example:

```
Ok
LIST
10 PRINT "Programming",
20 PRINT "is fun ",
30 PRINT "with the IBM PCjr."
40 END
Ok
RUN
Programming   is fun
with the IBM PCjr.
Ok
```

In this instance, commas were used to suppress the CR/LF, so the entries were placed in separate print zones. If the commas had been replaced by semicolons, the items would have been placed one after the other, on the same line.

PRINT USING

In contrast to PRINT, PRINT USING allows you to specify exactly the desired spacing and formatting. The command has the following structure:

PRINT USING format string;expressions

The format string consists of the actual definition of the desired spacing. It contains any special characters which are to be included

with the output. The format string also reserves an area on the output line where the data will be displayed. It, in effect, operates as a print zone, the length of which you have specified. The size of the reserved area is determined by the characters in the format string.

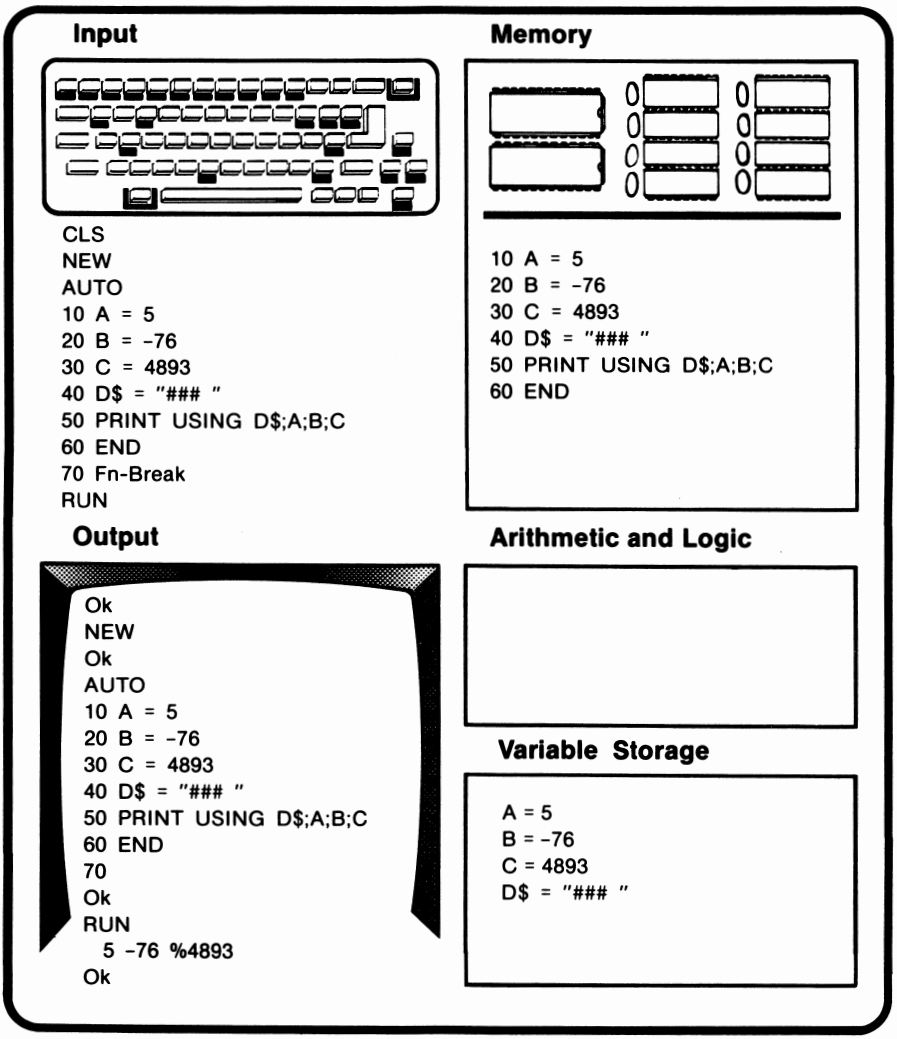
The format string must be followed by a semicolon, which separates it from the expressions. In this case, the expressions are the data to be output, often stored in variables. The individual data items must be delimited with commas or semicolons, but it doesn't matter which serves as the delimiter, because spacing is not affected.

Formatting Characters

There are two types of formatting characters, numeric and string. Numeric formatting characters work with numbers, and string formatting characters are used with strings. We will discuss the characters individually.

Numeric Formatting Characters -- Pound Sign (#)

The pound sign (#) is the most commonly used numeric formatting character. The pound sign is used to save a place for a digit. Each # reserves one space. This can be seen in the following example:



In this example a trailing space was used in D$ so that the numbers would be separated in the output. We will continue to follow this procedure.

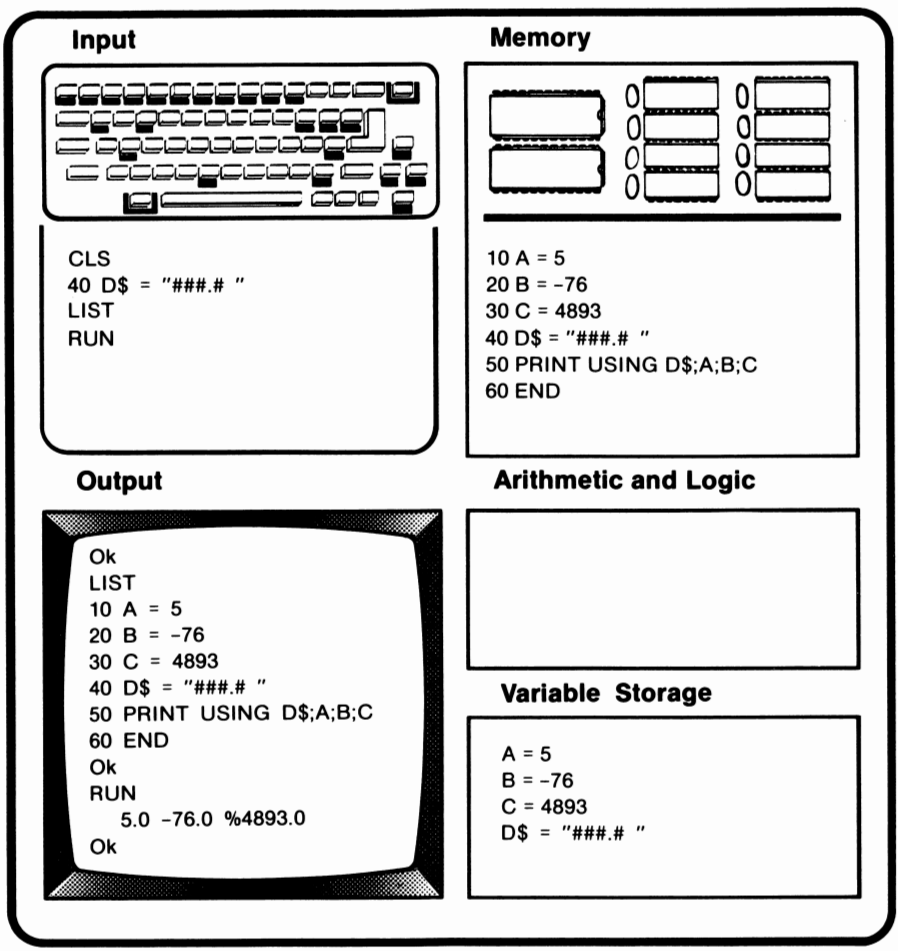
In addition, when this program was run, 4893 was preceded by %. The percent sign indicates that the number contains too many digits to fit into the allotted space. The % could be eliminated by inserting an additional #.

The negative sign in B occupies one of the digit spaces, so no

empty spaces were present in the field. If empty spaces were present, as is the case with A, the number would be right-justified. In other words, the field would be padded with blank spaces to the left of the number.

Numeric Formatting Characters -- Decimal Point (.)

The decimal point (.) can be placed anywhere within the string of formatting characters. Once set in place, it does not float. Notice what happens when line 40 is changed to include a decimal point:

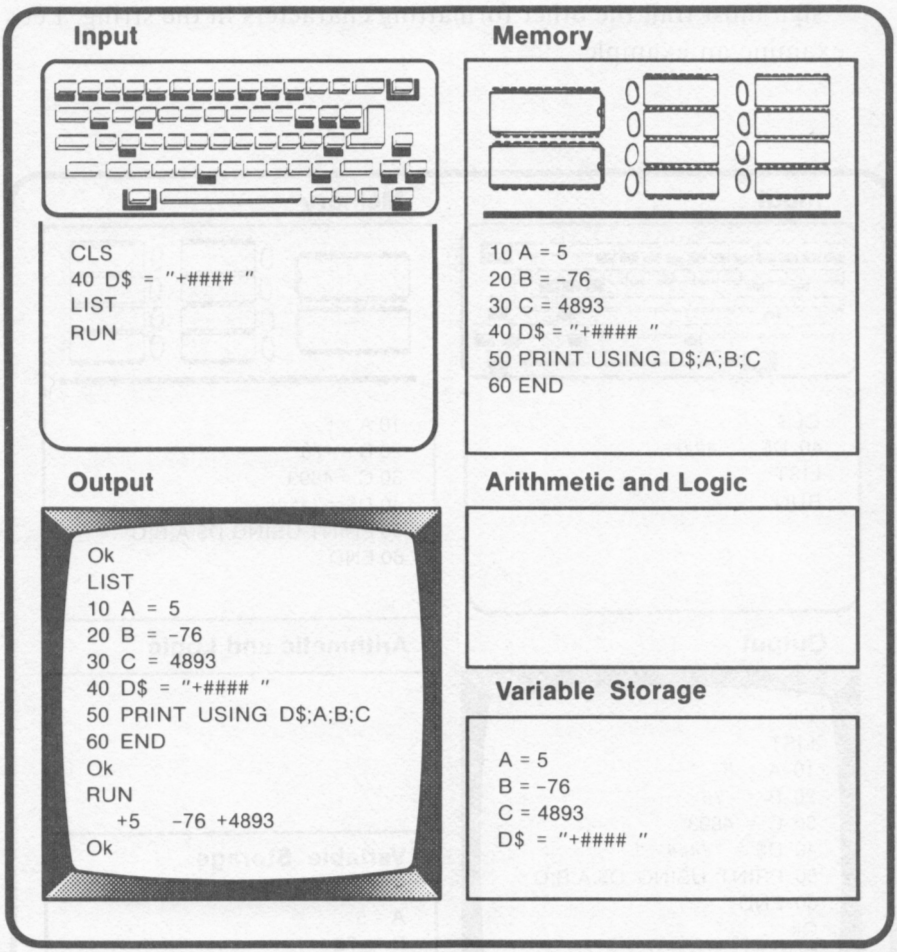


Again, there is a % before 4893.0, although an extra space was allotted, because the space was allotted after the decimal point. Under this structure, the number requires seven spaces, but only six are available.

In this example, all of the numbers were integers. If any of the numbers had been fixed-point numbers, the decimal portion would have been rounded to fit the allotted space. If a fixed-point number lacks a whole number portion, a 0 is placed prior to the decimal point. The only variation to this procedure occurs when no # appears before the decimal point. In this case, the zero will be omitted.

Numeric Formatting Characters -- Plus Sign (+)

The plus sign can be placed on either side of the #. The + sign causes the sign of the number to be printed next to the number. If the + is placed to the left of the #, the sign will float next to the number, on its left. If the + is placed to the right of the #, it will appear in the output in the space to the right of the number. When the + is used, it reserves an additional space for the sign. This can be seen in the following example:

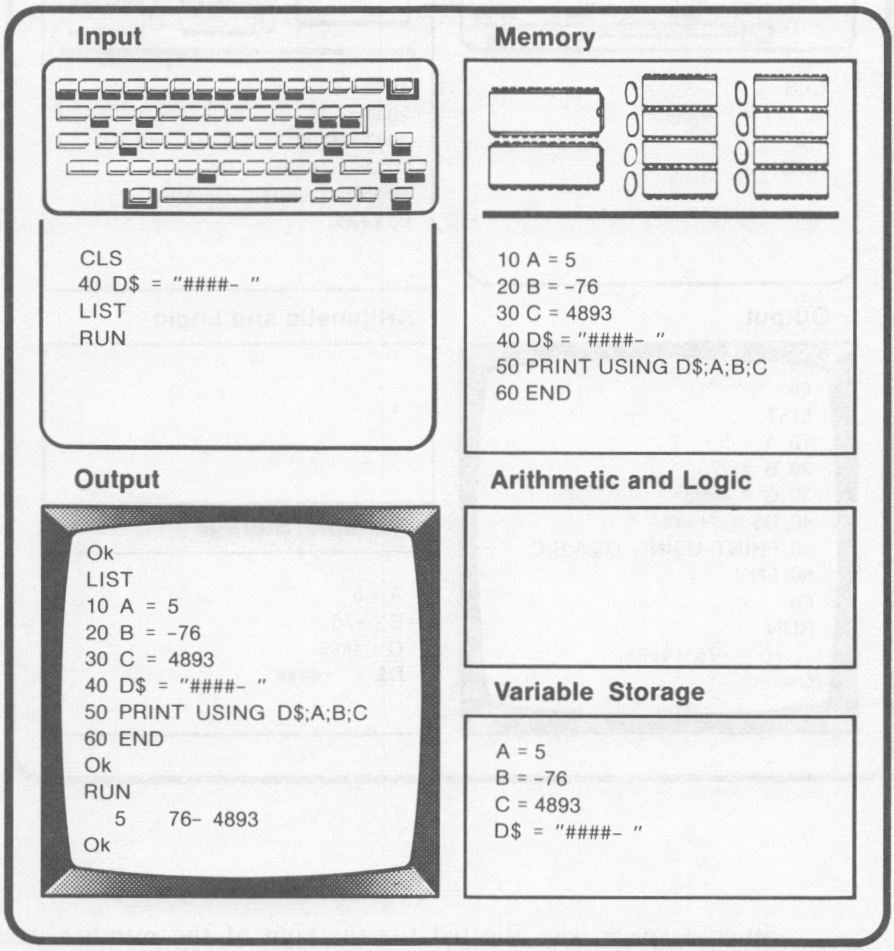


Since a space was allotted for the sign of the number, the negative sign in B no longer takes up a digit space.

Numeric Formatting Characters -- Minus Sign (-)

The minus sign functions similarly to +, except that the sign of the number only appears in the output if that number is negative. The

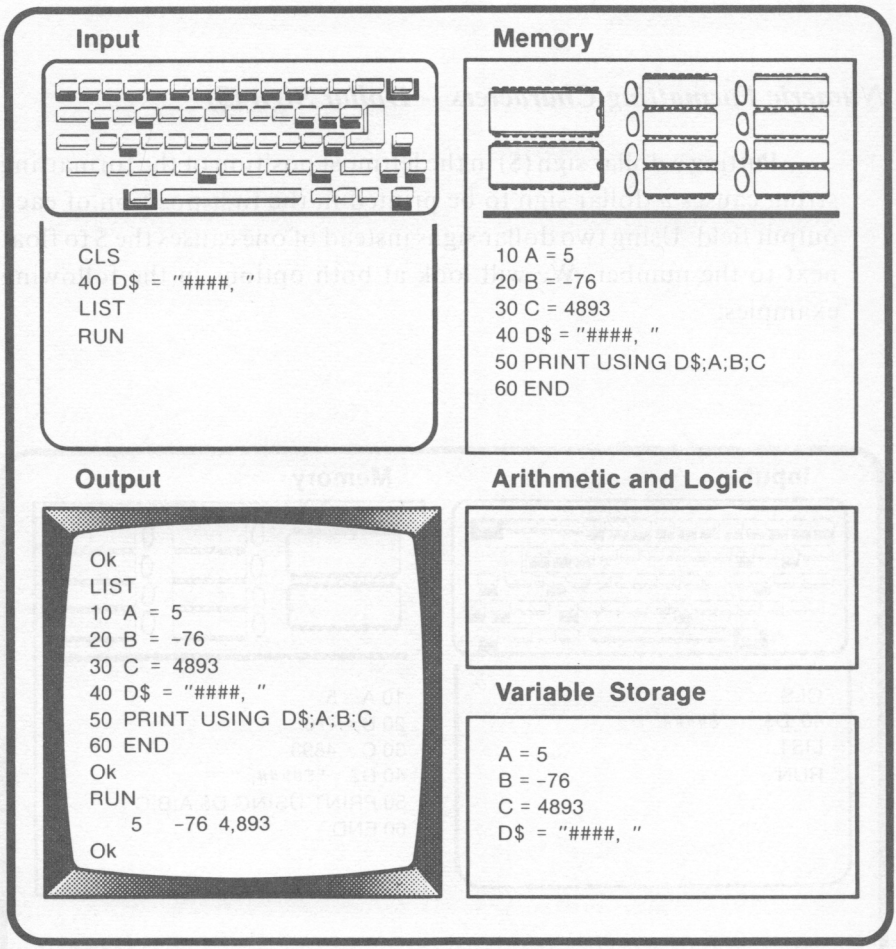
– sign must trail the other formatting characters in the string. Let's examine an example:



In this example, an additional space was reserved after the digits. When B was printed, the – took up that additional space. When the numbers are positive or zero, that space is left blank.

Numeric Formatting Characters -- Comma (,)

Commas can be placed anywhere within the string. The use of the comma is illustrated in this example:

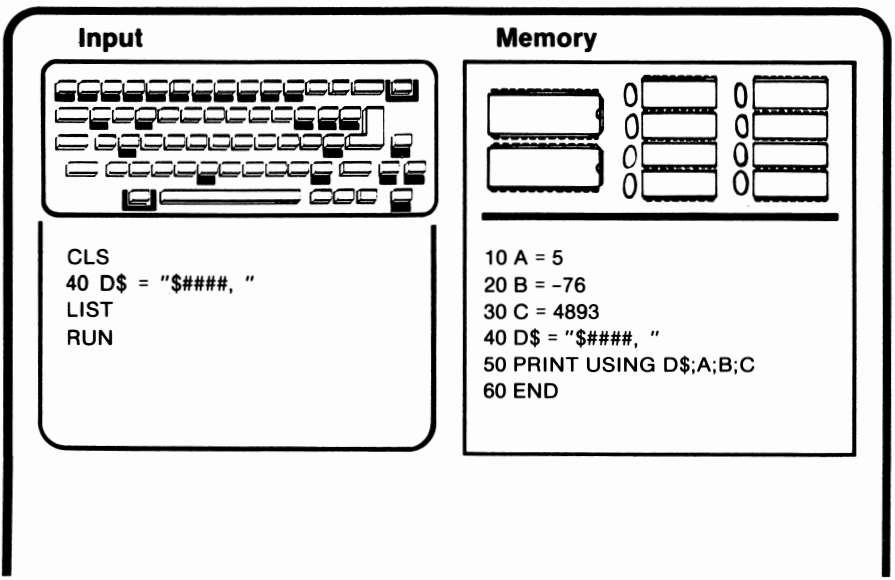


Notice that the comma reserved an additional space. When it wasn't needed, that space was taken by a blank. As many commas as desired can be placed in the format string, but the inclusion of just

one comma results in even the longest numbers having the correct number of commas inserted. When specifying a comma in a format string, remember that each comma will require a space. Therefore, additional #'s must be included in the format string to allow for the extra commas.

Numeric Formatting Characters -- Dollar Sign ($)

Putting a dollar sign ($) in the leftmost position of the formatting string causes a dollar sign to be printed in the first position of each output field. Using two dollar signs instead of one causes the $ to float next to the number. We will look at both options in the following examples:



Output

```
Ok
NEW
Ok
AUTO
 10 FOR FLOOR = 1 TO 2
 20 FOR ROOM = 1 TO 5
 30 INPUT "Resident";APT$
 (FLOOR,ROOM)
 40 NEXT ROOM
 50 NEXT FLOOR
 60 FOR A = 1 TO 2
 70 FOR B = 1 TO 5
 80 PRINT "Resident of";A;B;"is  ";
 APT$(A,B)
 90 NEXT B
100 NEXT A
110 END
120
Ok
RUN
Resident? Adams
Resident? Bell
Resident? Clark
Resident? Drake
Resident? Evans
Resident? Fletcher
Resident? Grant
Resident? Harris
Resident? Jacobs
Resident? Keller
Resident of 1   1 is Adams
Resident of 1   2 is Bell
Resident of 1   3 is Clark
Resident of 1   4 is Drake
Resident of 1   5 is Evans
Resident of 2   1 is Fletcher
Resident of 2   2 is Grant
Resident of 2   3 is Harris
Resident of 2   4 is Jacobs
Resident of 2   5 is Keller
Ok
```

Arithmetic and Logic

```
1 + 1 = 2     2 + 1 = 3     3 + 1 = 4
4 + 1 = 5     5 + 1 = 6     1 + 1 = 2
1 + 1 = 2     2 + 1 = 3     3 + 1 = 4
4 + 1 = 5     5 + 1 = 6     2 + 1 = 3
1 + 1 = 2     2 + 1 = 3     3 + 1 = 4
4 + 1 = 5     5 + 1 = 6     1 + 1 = 2
1 + 1 = 2     2 + 1 = 3     3 + 1 = 4
4 + 1 = 5     5 + 1 = 6     2 + 1 = 3
```

Variable Storage

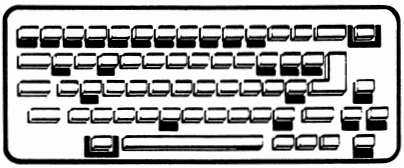
```
FLOOR = 1   ROOM = 1
APT$(1,1) = "Adams"
ROOM = 2  APT$(1,2) = "Bell"
ROOM = 3  APT$(1,3) = "Clark"
ROOM = 4  APT$(1,4) = "Drake"
ROOM = 5  APT$(1,5) = "Evans"
ROOM = 6   FLOOR = 2
ROOM = 1  APT$(2,1) = "Fletcher"
ROOM = 2  APT$(2,2) = "Grant"
ROOM = 3  APT$(2,3) = "Harris"
ROOM = 4  APT$(2,4) = "Jacobs"
ROOM = 5  APT$(2,5) = "Keller"
ROOM = 6   FLOOR = 3
A = 1      B = 1      B = 2
B = 3      B = 4      B = 5
B = 6      A = 2      B = 1
B = 2      B = 3      B = 4
B = 5      B = 6      A = 3
```

The data structure used in this program is a **table.** A table has two dimensions. The following diagram illustrates the structure of the table:

Tables

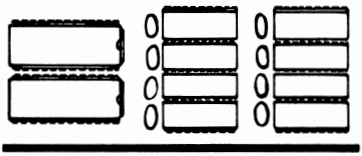
In the previous example, the apartment building had one floor. If the apartment building had contained more than one floor, we could have simply added numbers to our loop and assumed that we knew on which floor the apartments were. We could, however, modify the previous program so that each apartment was identified by both the floor and the room. This modified program is illustrated in the following example:

Input



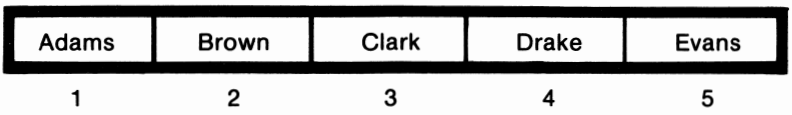
```
CLS
NEW
AUTO
 10 FOR FLOOR = 1 TO 2
 20 FOR ROOM = 1 TO 5
 30 INPUT "Resident";APT$
 (FLOOR,ROOM)
 40 NEXT ROOM
 50 NEXT FLOOR
 60 FOR A = 1 TO 2
 70 FOR B = 1 TO 5
 80 PRINT "Resident of";A;B;"is  ";
 APT$(A,B)
 90 NEXT B
100 NEXT A
110 END
120 Fn-Break
RUN
Adams
Bell
Clark
Drake
Evans
Fletcher
Grant
Harris
Jacobs
Keller
```

Memory

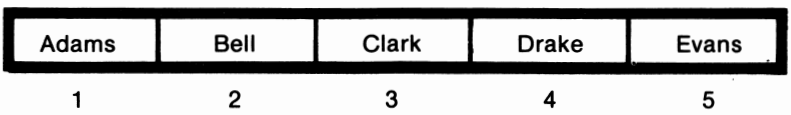


```
 10 FOR FLOOR = 1 TO 2
 20 FOR ROOM = 1 TO 5
 30 INPUT "Resident";APT$
(FLOOR,ROOM)
 40 NEXT ROOM
 50 NEXT FLOOR
 60 FOR A = 1 TO 2
 70 FOR B = 1 TO 5
 80 PRINT "Resident of";A;B;"is  ";
 APT$(A,B)
 90 NEXT B
100 NEXT A
110 END
```

Each subscripted variable represents one apartment. In this case, the apartment building has one floor, so each room is identified with one number. This type of structure is known as an **array.** The following diagram illustrates the structure of an array:



Brown lives in APT$(2). If Brown moves out and Bell moves in, the array has this structure:

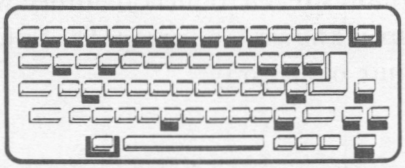


This array is identical to its predecessor except that one member, or element, has changed.

Each subscripted variable is a unique variable. In other words, APT$(1) does not refer to the same place in memory as APT$(2). One element in the array can be changed without affecting other elements in the array. It is also important to note that subscripted variables can be assigned values or used as operands. Any action which can be performed with numeric variables can be performed with subscripted numeric variables. Any action which can be performed with string variables can be performed with subscripted string variables.

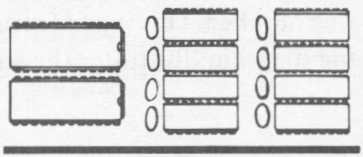
Finally, notice that we used FOR, NEXT loops in the program. The index variable acted both as a counter and as the subscript.

Input



```
CLS
NEW
AUTO
10 FOR ROOM = 1 TO 5
20 INPUT "Resident";APT$(ROOM)
30 NEXT ROOM
40 FOR A = 1 TO 5
50 PRINT "Resident of";A;"is  ";
     APT$(A)
60 NEXT A
70 END
80 Fn-Break
RUN
Adams
Brown
Clark
Drake
Evans
```

Memory



```
10 FOR ROOM = 1 TO 5
20 INPUT "Resident";APT$(ROOM)
30 NEXT ROOM
40 FOR A = 1 TO 5
50 PRINT "Resident of";A;"is  ";
     APT$(A)
60 NEXT A
70 END
```

Output

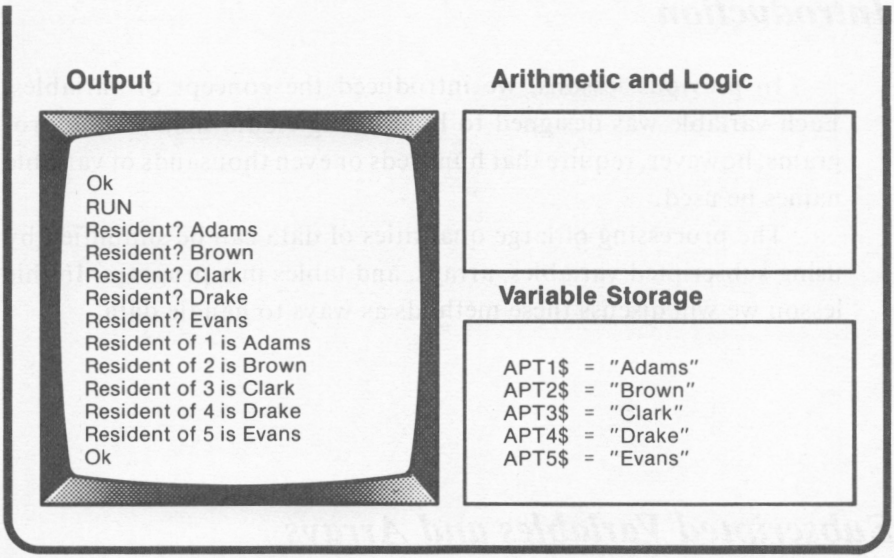
```
Ok
AUTO
10 FOR ROOM = 1 TO 5
20 INPUT "Resident";APT$(ROOM)
30 NEXT ROOM
40 FOR A = 1 TO 5
50 PRINT "Resident of";A;"is  ";
     APT$(A)
60 NEXT A
70 END
80
Ok
RUN
Resident? Adams
Resident? Brown
Resident? Clark
Resident? Drake
Resident? Evans
Resident of 1 is Adams
Resident of 2 is Brown
Resident of 3 is Clark
Resident of 4 is Drake
Resident of 5 is Evans
Ok
```

Arithmetic and Logic

```
1 + 1 = 2    2 + 1 = 3    3 + 1 = 4
4 + 1 = 5    5 + 1 = 6    1 + 1 = 2
2 + 1 = 3    3 + 1 = 4    4 + 1 = 5
5 + 1 = 6
```

Variable Storage

```
ROOM = 1     APT$(1) = "Adams"
ROOM = 2     APT$(2) = "Brown"
ROOM = 3     APT$(3) = "Clark"
ROOM = 4     APT$(4) = "Drake"
ROOM = 5     APT$(5) = "Evans"
ROOM = 6     A = 1
A = 2        A = 3
A = 4        A = 5
A = 6
```



A separate INPUT statement was required to input each resident's name. This simple example required 5 INPUT statements, 5 PRINT statements, and 5 variables. If the building had included 15 apartments, the program would have needed 15 INPUT statements, 15 PRINT statements, and 15 variable names. Imagine the length of the program if the apartment building had 50 residents! As the number of residents increases, the length of the program balloons.

This program can be written using another option, **subscripted variables.** A subscripted variable is a variable which includes a **subscript.** A subscript is a number in parentheses which follows the variable name. C(7) is an example of a subscripted variable with 7 as its subscript.

The use of subscripted variables helps make programming easier and programs shorter. The following program uses subscripted variables:

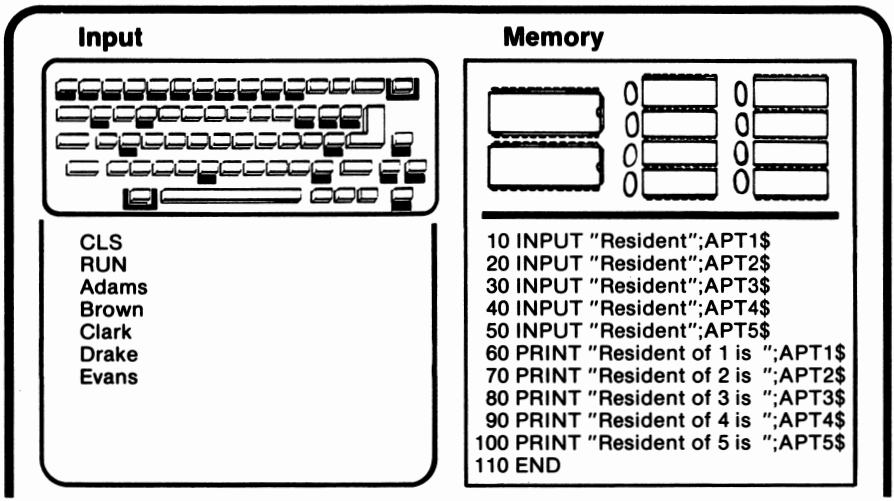
Introduction

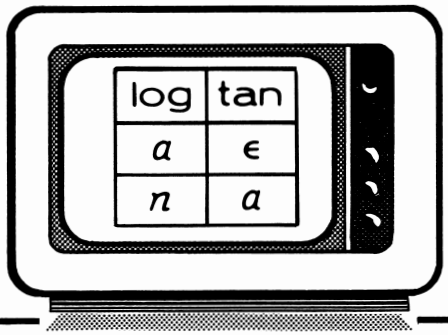
In previous lessons, we introduced the concept of variables. Each variable was designed to hold a single data item. Some programs, however, require that hundreds or even thousands of variable names be used.

The processing of large quantities of data can be simplified by using subscripted variables, arrays, and tables in a program. In this lesson we will discuss these methods as ways to handle data.

Subscripted Variables and Arrays

Suppose we want to input the names of the 5 residents of an apartment building and then receive a list of the apartment number and name of each resident. The following program illustrates the use of many individual variable names to accomplish this task:





Tables and Arrays

lesson 16

Lesson Goals

- *Learn what subscripted variables are and how to use them*
- *Learn what arrays and tables are and how to use them*
- *Learn when and how to use DIM*
- *Learn how to use OPTION BASE*
- *Learn how to use DATA and READ*
- *Learn how to use ERASE*

b. Print out whether the month is in the winter (1-3), spring (4-6), summer (7-9), or fall (10-12).

c. Using GOSUB, check to see if the month is within the specified range of 1 to 12. If not, have the user reenter the data.

d. Set up the program so that the user can input 5 months. Use either FOR, NEXT or WHILE, WEND.

2. What will be the value of A when WHILE A < 10 has finished executing?

   A. 10
   B. 11
   C. 0
   D. 1
   E. None of the above

3. Which of the following symbols will allow the same line number to be assigned to two statements?

   A. ;
   B. ,
   C. :
   D. !
   E. None of the above

4. Which of the following locations is the best place for a subroutine?

   A. The line after the GOSUB
   B. The line after the GOTO
   C. In an IF THEN ELSE as part of the ELSE
   D. After the END
   E. None of the above

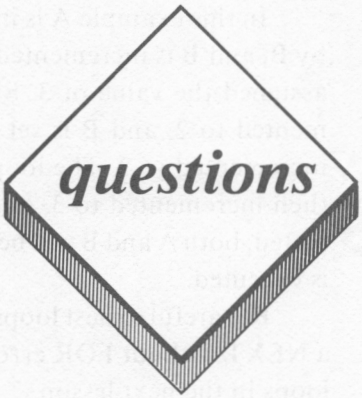
5. Which of the following commands will stop the execution of an infinite loop?

   A. Fn-Break
   B. Space Bar
   C. Enter
   D. Escape
   E. None of the above

Computer Exercises

1. Write a program to do the following:

   a. Allow the user to input a month, using the numbers 1 to 12. January should be represented by 1 and December by 12.



True or False

1. GOTO is used with subroutines.
2. Fn-Break will stop an infinite loop.
3. FOR loops can be nested.
4. The value of the index variable of WHILE should not be altered inside the loop, but the value of the index variable of FOR should be.
5. The ELSE part of an IF THEN executes only if the condition evaluates to true.

Multiple Choice

1. What will be the value of A when FOR A = 7 to 3 STEP -1 has finished executing?

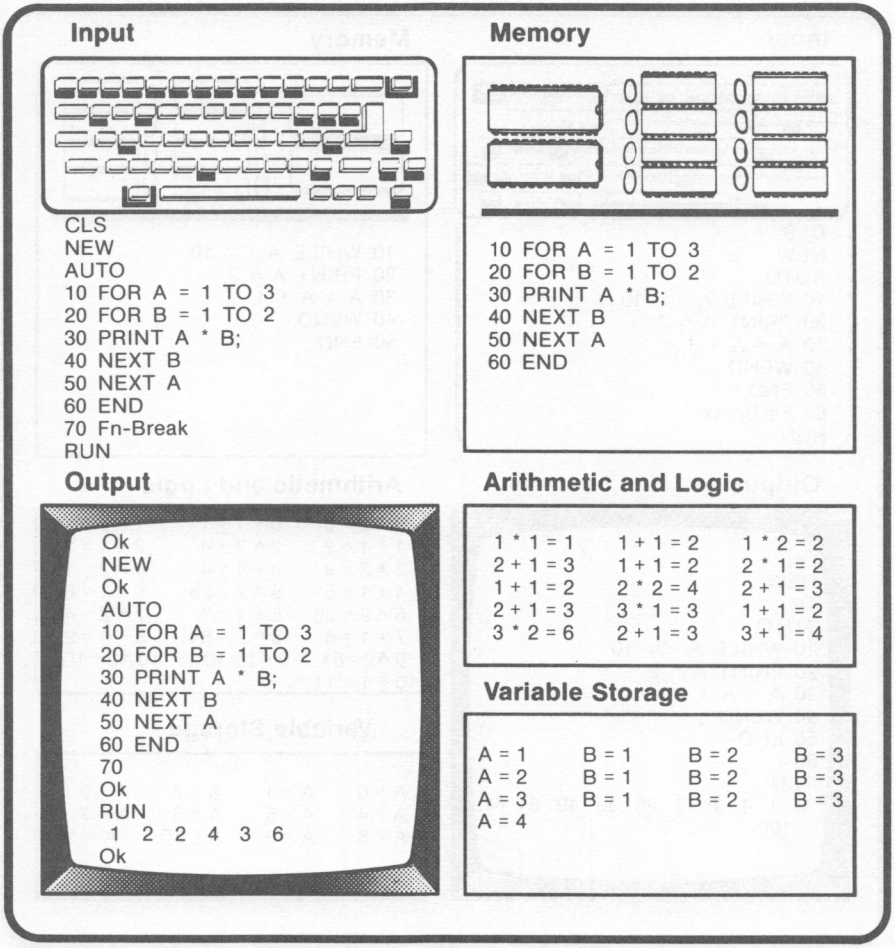
   A. -1
   B. 2
   C. 0
   D. 5
   E. None of the above

In this example A is initially set to 1 as is B. A is then multiplied by B, and B is incremented to 2. A is again multiplied by B, and B is assigned the value of 3. Since 3 is outside of B's range, A is incremented to 2, and B is set to 1. A and B are again multiplied. B is incremented to 2. The loop repeats, and B is incremented to 3. A is then incremented to 3, and B is reset to 1. When this cycle is completed, both A and B will be outside of their specified ranges, so line 60 is executed.

Be careful to nest loops properly. Loops which overlap will cause a NEXT without FOR error. We will discuss the uses of nested FOR loops in the next lesson.

In this example, the initial value of A is 0. If the index variable is not assigned a value prior to entering the loop, it is set by default to 0.

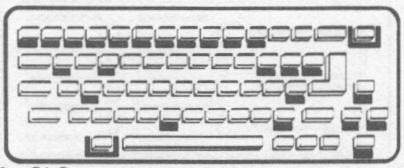
Both WHILE, WEND and FOR, NEXT loops can be **nested.** Nested means that one loop is contained entirely within the other. Let's examine the following example using FOR, NEXT loops:



WHILE, WEND

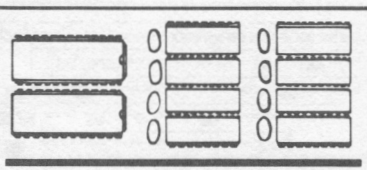
The WHILE, WEND statements can also be used to control loops. The loop will execute as long as WHILE evaluates as 0. Notice the following example:

Input



```
CLS
NEW
AUTO
10 WHILE A <= 10
20 PRINT A ^ 2;
30 A = A + 1
40 WEND
50 END
60 Fn-Break
RUN
```

Memory



```
10 WHILE A <= 10
20 PRINT A ^ 2;
30 A = A + 1
40 WEND
50 END
```

Output

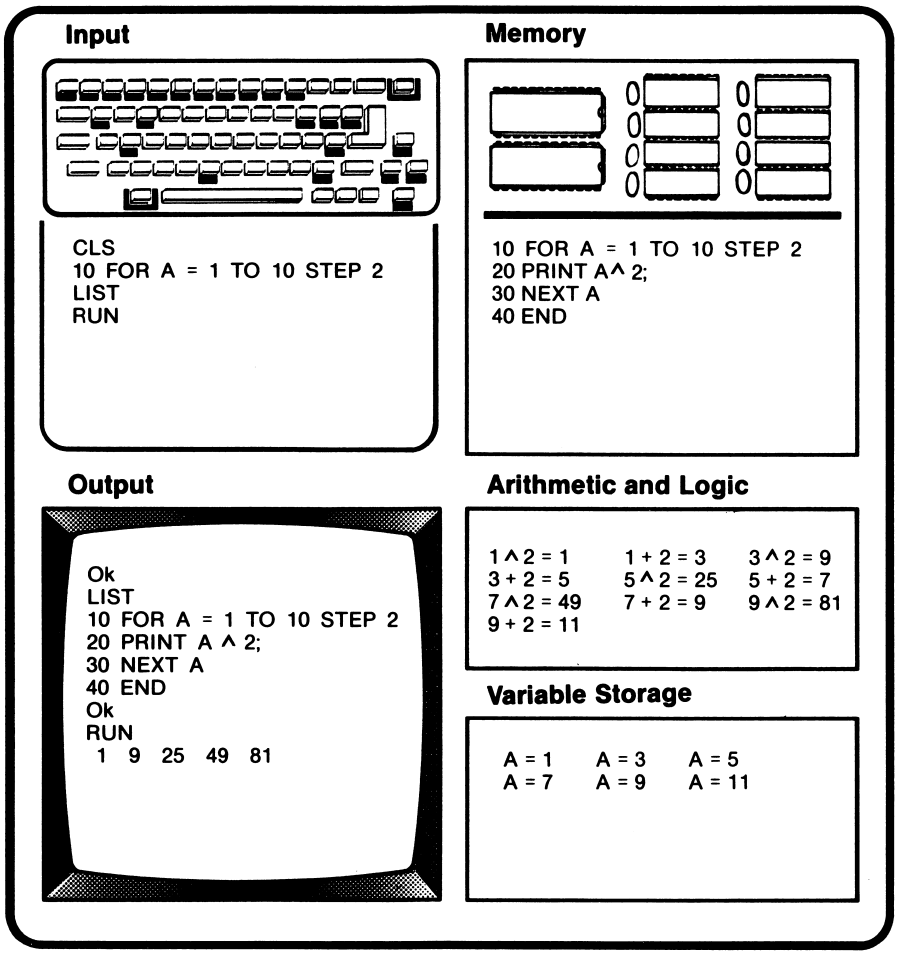
```
Ok
NEW
Ok
AUTO
10 WHILE A <= 10
20 PRINT A ^ 2;
30 A = A + 1
40 WEND
50 END
60
RUN
 0  1  4  9  16  25  36  49  64  81
  100
```

Arithmetic and Logic

```
0 ^ 2 = 0     0 + 1 = 1     1 ^ 2 = 1
1 + 1 = 2     2 ^ 2 = 4     2 + 1 = 3
3 ^ 2 = 9     3 + 1 = 4     4 ^ 2 = 16
4 + 1 = 5     5 ^ 2 = 25    5 + 1 = 6
6 ^ 2 = 36    6 + 1 = 7     7 ^ 2 = 49
7 + 1 = 8     8 ^ 2 = 64    8 + 1 = 9
9 ^ 2 = 81    9 + 1 = 10    10 ^ 2 = 100
10 + 1 = 11
```

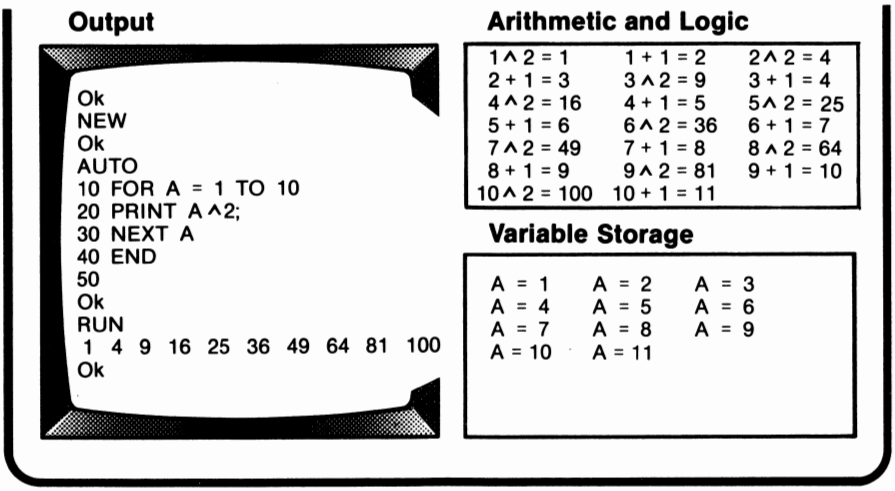
Variable Storage

```
A = 0    A = 1    A = 2    A = 3
A = 4    A = 5    A = 6    A = 7
A = 8    A = 9    A = 10   A = 11
```



The value for STEP need not necessarily be positive. A negative STEP value counts down. Using a STEP value of 0 creates an infinite loop.

Be careful not to change the value of the index variable within the loop, as an error may result.



Each time NEXT is executed, A is incremented by 1, and control transfers to line 10. A is checked to see if it is within the specified range, in this case, 1 to 10. The loop executes ten times. During the last execution, one is added to 10 when NEXT is executed. Control returns to line 10, but since $11 > 10$, execution skips to the line after NEXT, line 40. It is important to note that A = 11 when the loop is completed, not 10.

A is known as the **index variable.** Specification of the index variable is not mandatory after NEXT, but deletion of it can cause confusion.

In this example, the loop was incremented by 1 during each execution. Any number, integer or real, can be used. This number is specified with the word STEP. STEP designates the increment. The default value for STEP is 1. Let's examine the program after it has been modified to use Step 2.

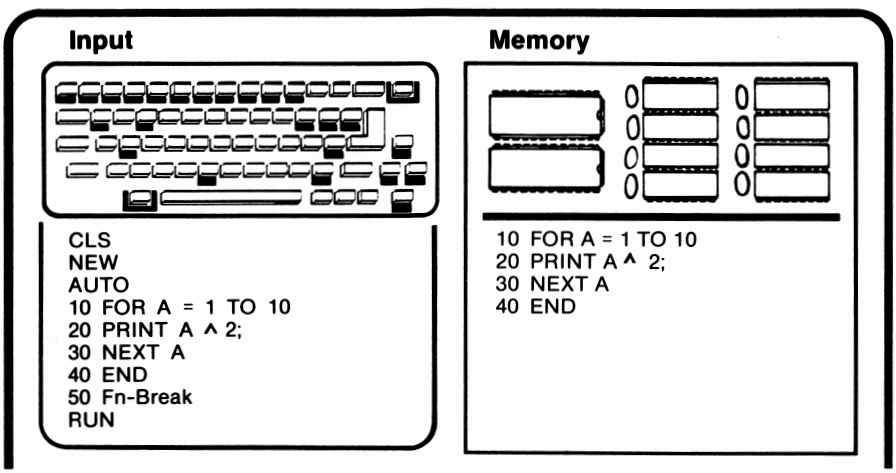
In this program, the colon was used to permit two statements on the same program line, line number 40. As shown in this example, the colon can be extremely useful when the desired result is to have two or more statements execute when a condition is true. The THEN statement is operative until the end of the line is reached or until ELSE is encountered. The maximum length for the line is still 255 characters.

Both IF THEN 10 and IF THEN GOTO 10 send control to line 10. Including the GOTO statement can limit confusion.

FOR, NEXT

Earlier in this lesson we used GOTO to repeat a portion of a program. The difficulty with using GOTO was that we could not tell it when to stop except with Fn-Break. No lines after GOTO were executed.

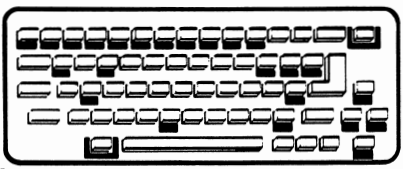
FOR, NEXT can be used to set up a portion of the program to repeat a finite number of times. This process of repetition is known as **looping.** The use of FOR, NEXT is illustrated in the following program:



Conditional Statements with Branching

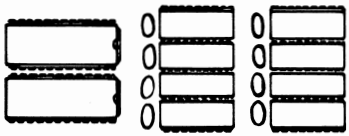
Branching can be used with conditional statements. Let's examine an example:

Input



```
CLS
LIST
RUN
40
36
```

Memory



```
10 INPUT "Number of wins";WINS
20 INPUT "Number of losses";
   LOSSES
30 PERCENT = WINS / (WINS +
   LOSSES)
40 IF PERCENT > .5 THEN PRINT
   "The team has a winning record.":
   GOTO 60
50 IF PERCENT = .5 THEN PRINT
   "The team has as many wins as
   losses." ELSE PRINT "The team
   has a losing record."
60 END
```

Output

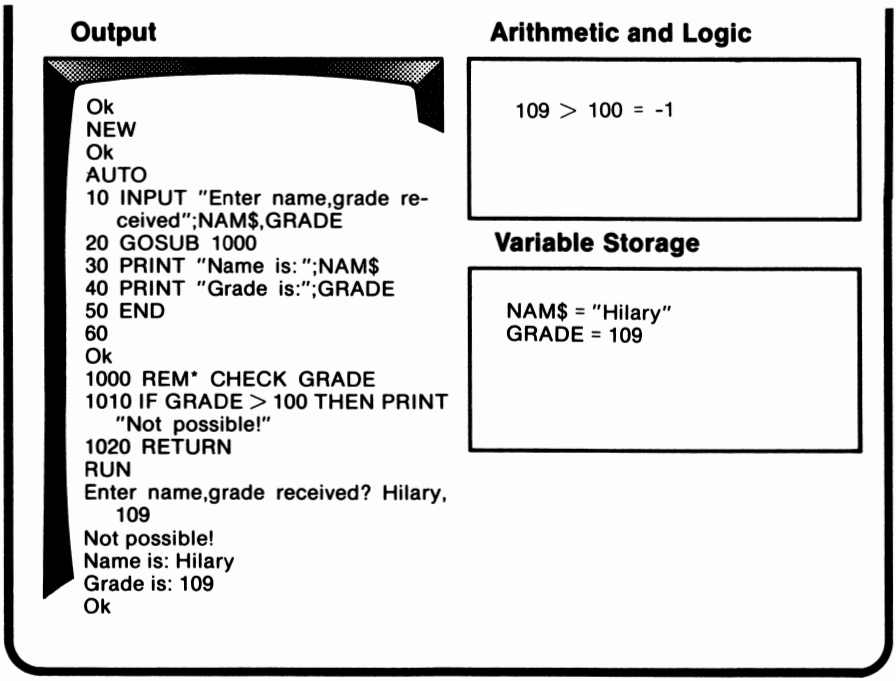
```
Ok
LIST
10 INPUT "Number of wins";WINS
20 INPUT "Number of losses";
   LOSSES
30 PERCENT = WINS / (WINS +
   LOSSES)
40 IF PERCENT > .5 THEN PRINT
   "The team has a winning record.":
   GOTO 60
50 IF PERCENT = .5 THEN PRINT
   "The team has as many wins as
   losses." ELSE PRINT "The team
   has a losing record."
60 END
Ok
RUN
Number of wins? 40
Number of losses? 36
The team has a winning record.
Ok
```

Arithmetic and Logic

```
40 / 76 = .5263158
.5263158 > .5 = -1
```

Variable Storage

```
WINS = 40
LOSSES = 36
PERCENT = .5263158
```



When GOSUB is encountered in line 20, the statement is executed and control shifts to line 1000. Execution continues sequentially until the RETURN statement is encountered in line 1020. RETURN causes execution to go to the line after the GOSUB, line 30. Execution is completed at line 50. If END is omitted, the subroutine executes again. When line 1020 executes this time, an error message, "RETURN without GOSUB in line 1020," appears.

Subroutines can make a program more efficient, and program writing is simplified by breaking a complex program into shorter segments. Subroutines are also easier to debug, because they are shorter.

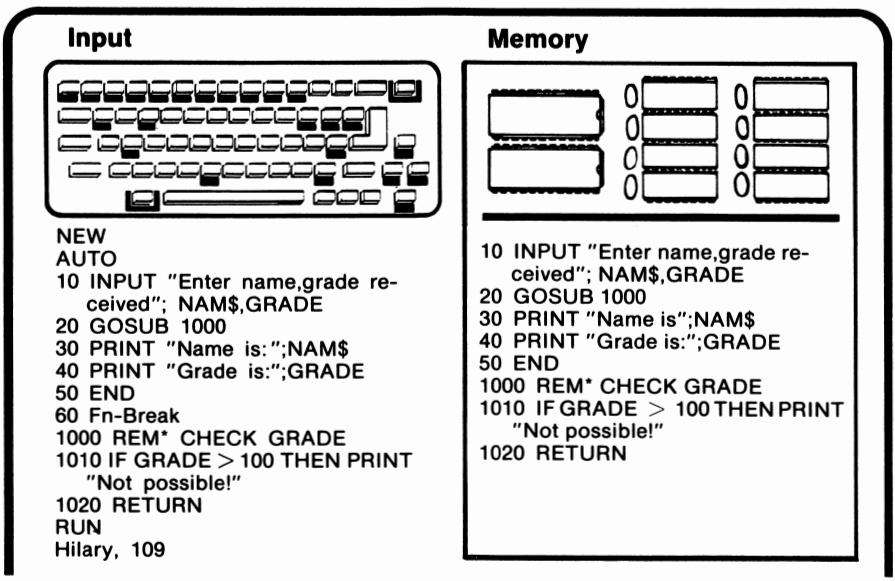
This program branches until Fn-Break is entered. Fn-Break interrupts execution. We will discuss other ways to control branching later in this lesson.

GOSUB, RETURN

A **subroutine** is a small program within a larger program. Subroutines allow you to use the same segment of a program over without having to reenter the individual program lines. There is no limit to the number of times a subroutine can be called.

GOSUB is the command which calls a subroutine. GOSUB is followed by the line number of the first line of the subroutine. Once execution has shifted to the subroutine, it continues normally until RETURN is encountered. RETURN shifts execution back to the main part of the program, specifically to the line after the GOSUB.

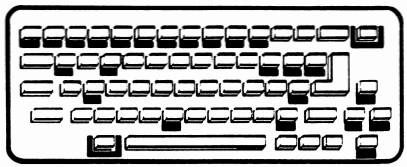
It is a good programming practice to identify each subroutine with a REM statement. Using REM makes it clear to others what the subroutine accomplishes. Another good idea is to group subroutines at the end of the program, after the END. This practice can reduce the chances for error.



GOTO

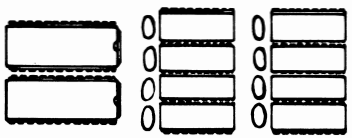
GOTO is a statement which changes the order of execution. GOTO is followed by a line number. After GOTO, the next statement executed is that line with the specified line number. Another word for this process is **branching.** The following example illustrates the use of GOTO:

Input



```
CLS
50 GOTO 10
LIST
RUN
33
32
33
34
Fn-Break
```

Memory



```
10 INPUT "Number of wins";WINS
20 INPUT "Number of losses";
   LOSSES
30 PERCENT = WINS / (WINS +
   LOSSES)
40 IF PERCENT >= .5 THEN PRINT
   "The team has at least as many
   wins as losses." ELSE PRINT
   "The team has a losing record."
50 GOTO 10
60 END
```

Output

```
Ok
LIST
10 INPUT "Number of wins";WINS
20 INPUT "Number of losses";
   LOSSES
30 PERCENT = WINS / (WINS +
   LOSSES)
40 IF PERCENT >= .5 THEN PRINT
   "The team has at least as many
   wins as losses." ELSE PRINT
   "The team has a losing record."
50 GOTO 10
60 END
Ok
RUN
Number of wins? 33
Number of losses? 32
The team has at least as many wins
as losses.
Number of wins? 33
Number of losses? 34
The team has a losing record.
Number of wins?
Break in 10
Ok
```

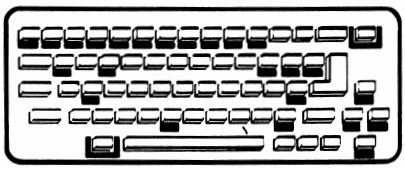
Arithmetic and Logic

```
33 / 65 = .5076924
.5076924 >= .5 = -1
33 / 67 = .4925373
.4925373 >= .5 = 0
```

Variable Storage

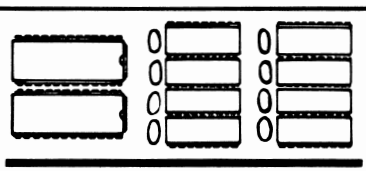
```
WINS = 33      LOSSES = 32
PERCENT = .5076924
WINS = 33      LOSSES = 34
PERCENT = .4925373
```

Input



```
CLS
40 IF PERCENT >= .5 THEN PRINT
   "The team has at least as many
   wins as losses." ELSE PRINT
   "The team has a losing record."
50
LIST
RUN
35
40
```

Memory



```
10 INPUT "Number of wins";WINS
20 INPUT "Number of losses";
LOSSES
30 PERCENT = WINS/(WINS +
LOSSES)
40 IF PERCENT > .5 THEN PRINT
"The team has at least as many
wins as losses." ELSE PRINT
"The team has a losing record."
60 END
```

Output

```
Ok
LIST
10 INPUT "Number of wins";WINS
20 INPUT "Number of losses";
   LOSSES
30 PERCENT = WINS / (WINS +
   LOSSES)
40 IF PERCENT >= .5 THEN PRINT
   "The team has at least as many
   wins as losses." ELSE PRINT
   "The team has a losing record."
60 END
Ok
RUN
Number of wins? 35
Number of losses? 40
The team has a losing record.
```

Arithmetic and Logic

```
35 / 75 = .4666667
.4666667 >= .5 = 0
```

Variable Storage

```
WINS = 35
LOSSES = 40
PERCENT = .4666667
```

When line 40 was executed, the comparison returned a value of false. The statements following THEN were ignored. The statements following ELSE were executed. If the comparison had been true, the THEN commands would have been executed, and the ELSE commands ignored. Larry uses IF THEN ELSE logic when he decides that if it is raining, he will take his umbrella, otherwise he will take his sunglasses.

Output

```
Ok
NEW
Ok
AUTO
10 INPUT "Number of wins";WINS
20 INPUT "Number of losses";
   LOSSES
30 PERCENT = WINS / (WINS +
   LOSSES)
40 IF PERCENT >= .5 THEN PRINT
   "The team has at least as many
   as losses."
50 IF PERCENT < .5 THEN PRINT
   "The team has a losing record."
60 END
70
Ok
RUN
Number of wins? 35
Number of losses? 40
The team has a losing record.
Ok
```

Arithmetic and Logic

```
35 / 75 = .4666667
.4666667 >= .5 = 0
.4666667 < .5 = -1
```

Variable Storage

```
WINS = 35
LOSSES = 40
PERCENT = .4666667
```

When line 40 was executed, the comparison was evaluated. Since the comparison is false, the rest of line 40 was ignored. When line 50 was executed, its comparison returned a value of true, and the rest of that line was executed.

This program can be condensed by using a modification of IF THEN, IF THEN ELSE. Let's examine this modification:

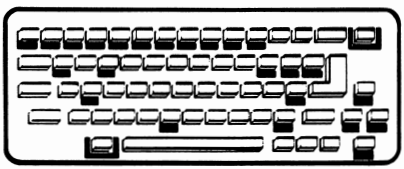
Introduction

In the past lessons, programs have been executed one line after the other, or sequentially. In this lesson we will discuss how to alter this pattern. We will look at how to make the program execute a statement only under certain conditions, how to make the program jump to another portion of the program, and how to make sections execute repeatedly.

IF THEN

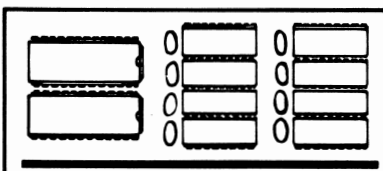
IF THEN sets up a check for a condition. If the condition is true, the commands following THEN are executed. If the condition is false, those commands are ignored. Larry uses IF THEN logic when he says that "if it is raining tomorrow, I will take my umbrella." The following program illustrates the use of IF THEN:

Input



```
CLS
NEW
AUTO
10 INPUT "Number of wins";WINS
20 INPUT "Number of losses";
   LOSSES
30 PERCENT = WINS / (WINS +
   LOSSES)
40 IF PERCENT >= .5 THEN PRINT
   "The team has at least as many
   wins as losses."
50 IF PERCENT < .5 THEN PRINT
   "The team has a losing record."
60 END
70 Fn-Break
RUN
35
40
```

Memory



```
10 INPUT "Number of wins";WINS
20 INPUT "Number of losses";
   LOSSES
30 PERCENT = WINS / (WINS +
   LOSSES)
40 IF PERCENT >= .5 THEN PRINT
   "The team has at least as many
   wins as losses."
50 IF PERCENT < .5 THEN PRINT
   "The team has a losing record."
60 END
```



Conditional, Branching, and Looping Statements

lesson 15

Lesson Goals

- *Learn how to use IF THEN to control execution of a program*
- *Learn how to use GOTO to alter the flow of execution*
- *Learn how to use subroutines and GOSUB*
- *Learn how to use conditional statements with branching*
- *Learn how to use FOR, NEXT statements for looping*
- *Learn how to use WHILE, WEND statements for looping*

2. What is the maximum number of characters which can be input using LINE INPUT?

   A. 255
   B. 32767
   C. It depends on what number is specified in the function call.
   D. 10
   E. None of the above

3. Which of the following symbols acts as a delimiter when INPUT is used with 2 or more variables?

   A. ;
   B. ,
   C. .
   D. INPUT will not accept more than one variable.
   E. None of the above

4. Which of the following is a function?

   A. INPUT
   B. INPUT$
   C. LINE INPUT
   D. All of the above
   E. None of the above

Computer Exercises

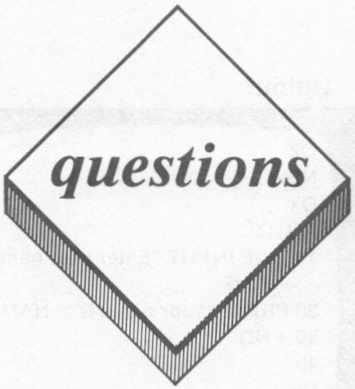
1. Rewrite the program you wrote for Computer Exercise 1 on page 128 to do the following:

   a. Input five student's names, letter grades, and percentages using INPUT.

   b. Allow the teacher to write a memo to him or herself in the program by using LINE INPUT.

   c. Input a ten-character teacher's name with INPUT$.

   Be sure to use prompts and to print out all data on the screen after it has been input.



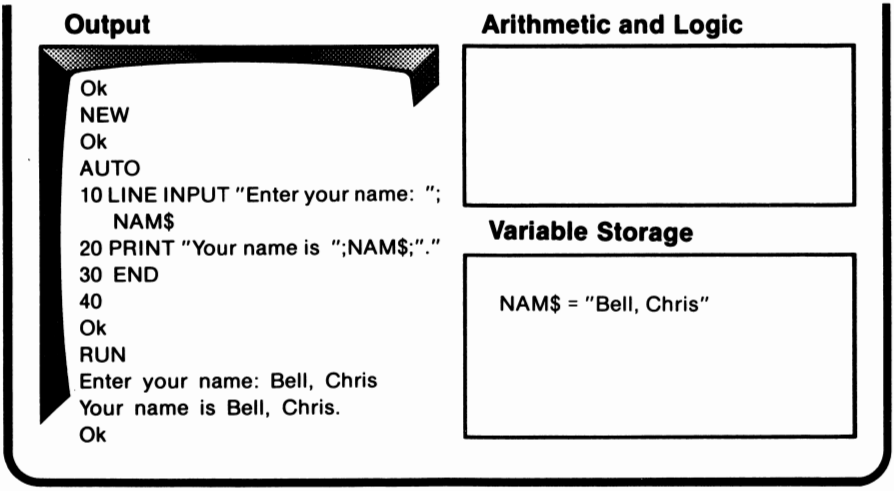
True or False

1. LINE INPUT works with numerics.
2. INPUT can be used to specify one or more entries.
3. A prompt is a message which tells the user what information to enter.
4. INPUT$ accepts any character, except Fn-Break.
5. LINE INPUT does not accept an empty string.

Multiple Choice

1. Which of the following characters appears on the screen when INPUT$ is used without a prompt?

   A. ?
   B. Flashing cursor
   C. Enter your name.
   D. Steady cursor
   E. None of the above



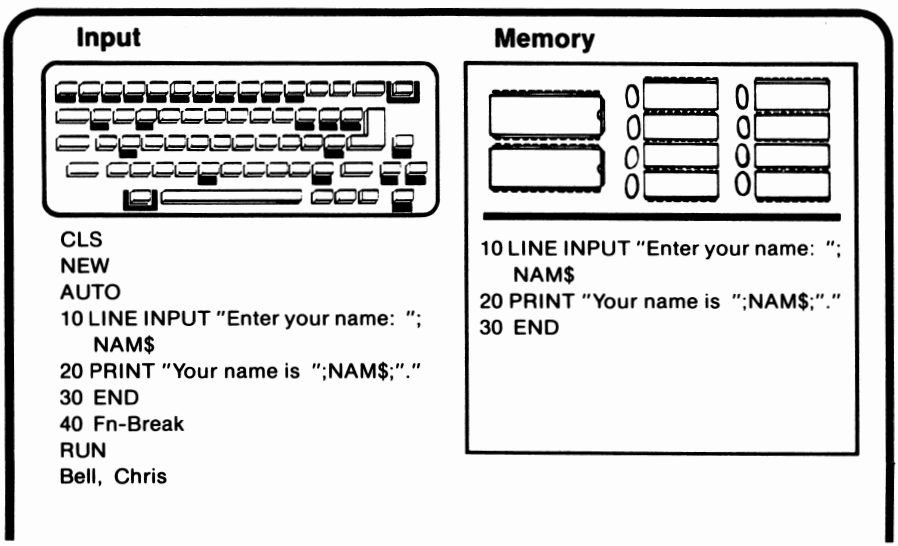
LINE INPUT and INPUT both assign an empty string to their variable if the Enter key is pressed prior to the inputting of any data.

Notice that the character that had been input did not appear on the screen until line 30 was executed. The Enter key was not pressed to signify the end of the data. The computer instead waited until 10 characters had been entered and then resumed execution.

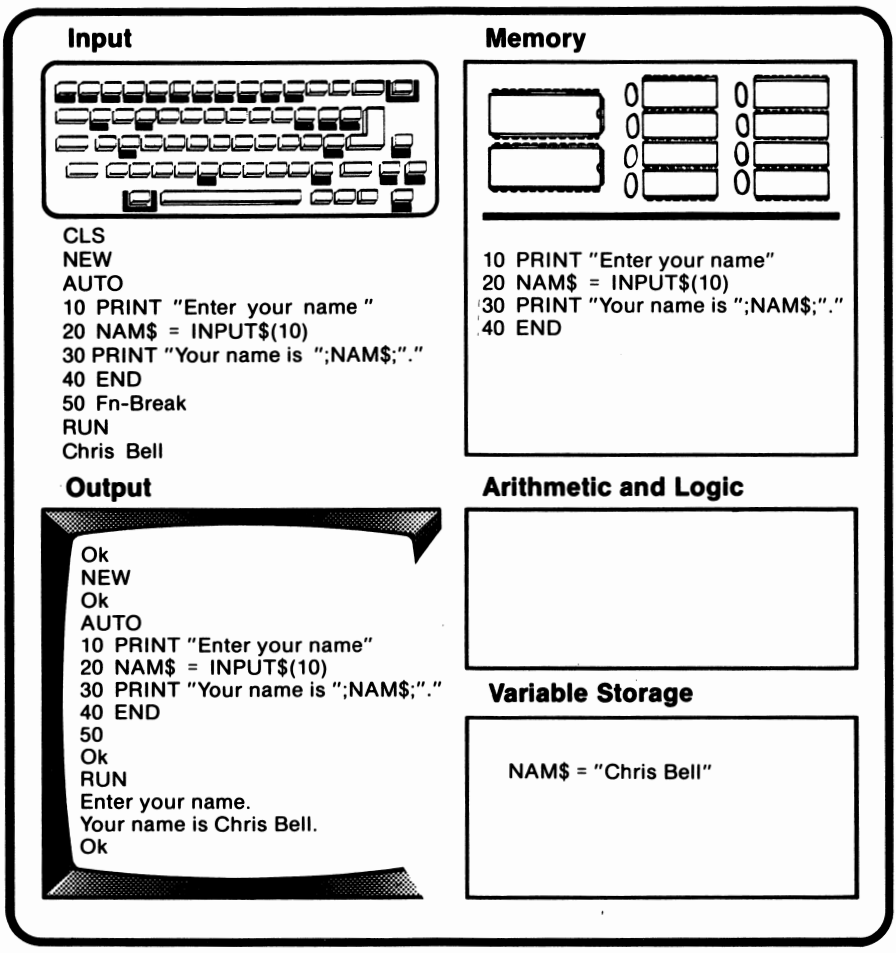
It is a good idea to include a prompt such as the message in line 10. Otherwise, the person using the program may not realize that data needs to be entered.

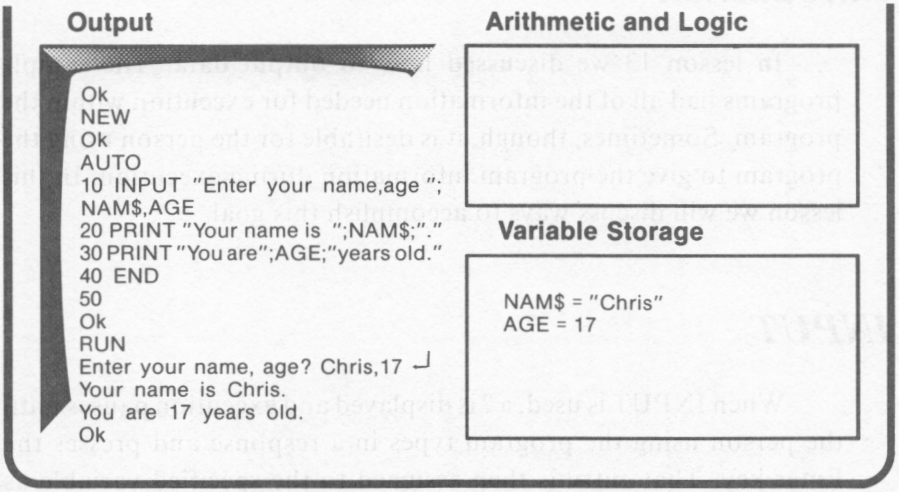
LINE INPUT

LINE INPUT allows a line of data to be input and assigned to a string variable. The number of characters does not have to be specified prior to execution, and up to 255 characters will be accepted. LINE INPUT accepts any character except Fn-Break. Unlike INPUT$, LINE INPUT prints a flashing cursor to indicate that the user should enter data. A prompt message may also be used. During execution, the program pauses for the input and resumes execution after the Enter Key has been pressed. Let's examine an illustration:



where a$ is a string variable and b is the number of characters to be accepted. The maximum value allowed for b is 255. Any character, including control characters, will be accepted. The only exception is Fn-Break. INPUT$ is often used when characters not accepted by INPUT, such as commas or quotation marks, are to be input. INPUT$ will only assign the input to string variables. The following program illustrates the use of INPUT$:





When the name and age were entered during execution of this program, the entries were separated by a comma. No blank spaces were entered. During execution, that comma is considered the delimiter of the separate entries.

Up to 255 characters will be accepted for a string. If the Enter key is pressed prior to the entry of any other characters, an empty string will be assigned to a string variable or 0 to a numeric variable. Finally, if the inputted string begins with a blank space or includes commas, it must be enclosed in quotes.

INPUT$

INPUT$ is a function which specifies the number of characters to be input and allows that number of characters to be entered. Its configuration is:

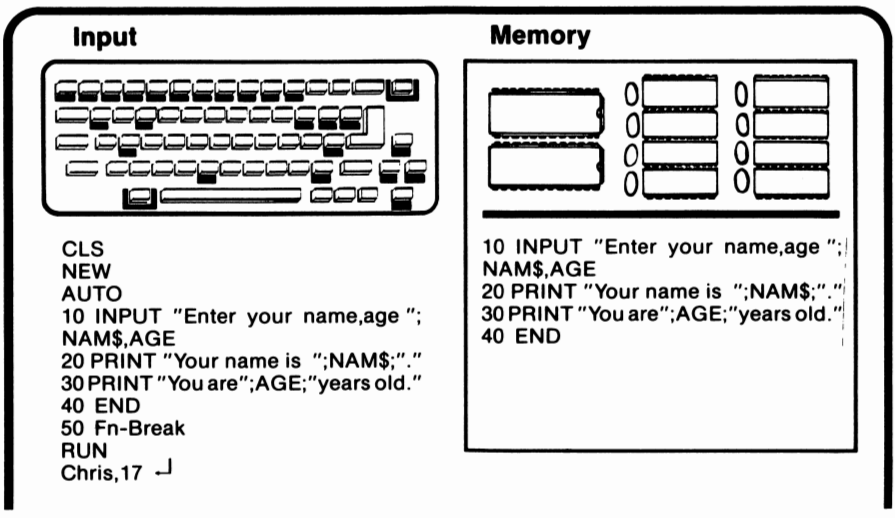
a$ = INPUT$(b)

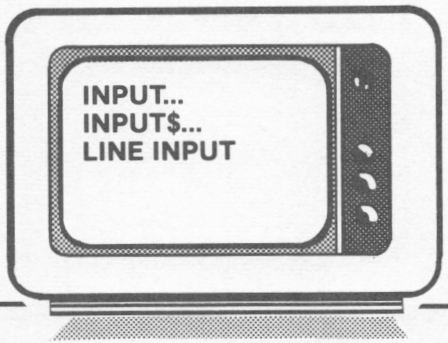
Introduction

In lesson 13 we discussed how to output data. The sample programs had all of the information needed for execution within the program. Sometimes, though, it is desirable for the person using the program to give the program information during execution. In this lesson we will discuss ways to accomplish this goal.

INPUT

When INPUT is used, a ? is displayed and execution pauses until the person using the program types in a response and presses the Enter key. That entry is then assigned to the specified variable as program execution resumes where it stopped. INPUT can be used to specify one or more entries of one or more variable types. Because of this possibility, it is a good idea to **prompt** the user. A prompt is a message which tells the person using the program what information to enter. A typical prompt is "Type END to end program." If INPUT is used for more than one entry in the same line, the values will be assigned to the variables in the order listed. Let's examine an example:





Inputting Data

lesson 14

Lesson Goals

- *Learn how to input data using INPUT*
- *Learn how to use the INPUT$ function*
- *Learn how to use LINE INPUT with and without prompts*

2. Assuming the screen width is set to 40, which of the following PRINT commands will not cause the output to be placed on one line?

   A. PRINT "Ann" "Bob" "Chris"
   B. PRINT "Ann";"Bob";"Chris"
   C. PRINT "Ann","Bob","Chris"
   D. PRINT "Ann""Bob""Chris"
   E. None of the above

3. What would be the output of the following line:

   PRINT USING "&";500;-9353;1

   A. It wouldn't work
   B. 500-9353 1
   C. %500%-93531
   D. 50093531
   E. None of the above

4. What does % mean?

   A. It saves a place for a digit.
   B. It saves a place for a character.
   C. When used in numeric formatting, % indicates percentage.
   D. It means the number which was sent to the field was too long for the field.
   E. None of the above

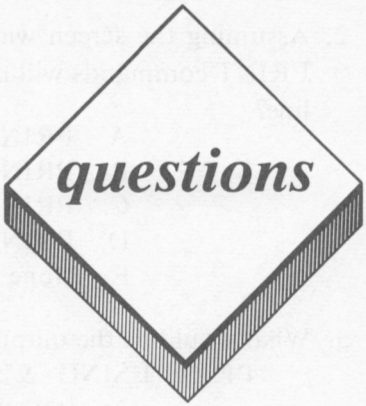
Computer Exercises

1. Revise the program you wrote for Computer Exercise 1 on page 128 to do the following:

   a. Use the PRINT statement to print the output. Use a separate string for the name, percentage, and grade of each student. The data for each student should be input on the same line.

   b. Use PRINT USING to print the data. Treat the names and letter grades as strings, and the percentages as numerics.

   Make sure the data is still centered under its proper heading.



True or False

1. The # is used to save a place for a character in string formatting.
2. When ** is used, in order for the $ to float, $$ must be specified.
3. The & has no special meaning in numeric formatting.
4. If a string is shorter than the field allots, it will be left-justified.
5. SPC can be used with PRINT USING.

Multiple Choice

1. Which of the following formatting characters is used when formatting strings?

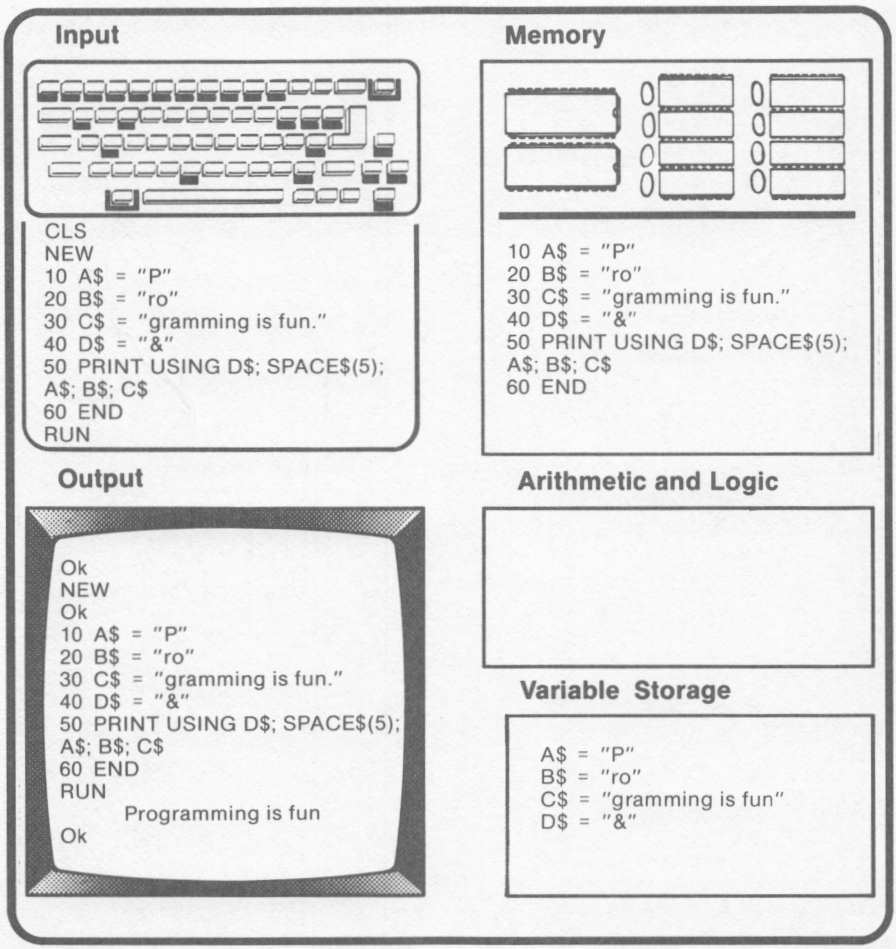
   A. +
   B. !
   C. ^^^^
   D. ,
   E. None of the above

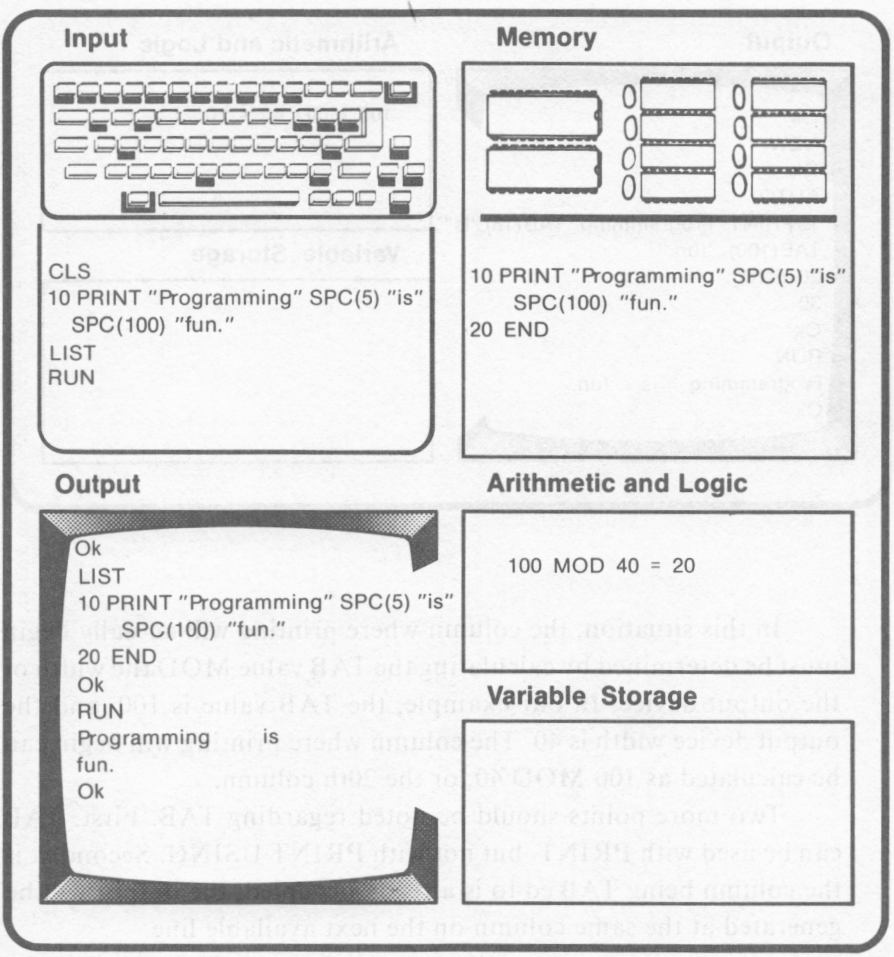
An important point to note is that all of these formatting functions can be called with variables. It is not necessary to use them only with constants.

output position is fixed. TAB(15) results in data being printed in the fifteenth column either on that line, if the space is available, or on the next. SPC(15) results in data appearing 15 spaces after the last item.

SPACE$

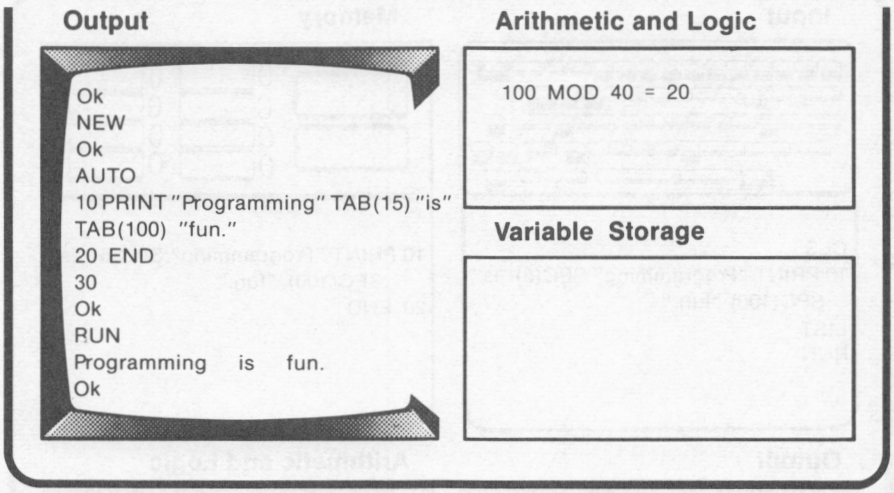
SPACE$ returns a string consisting of the number of spaces given to it when it is called. SPACE$ can be used with PRINT USING. This is illustrated in the following example:





As with TAB, if the value given to SPC is larger than the output device width, the number of spaces inserted will equal the given value MOD the output width. If necessary, output will be printed on the next line in column 0 in order to obtain the proper spacing. SPC cannot be used with PRINT USING.

Since the position in which the output will be printed with SPC is dependent upon the length of previous strings, TAB is the more frequent choice when tables need to be set up. This is because the



In this situation, the column where printing will actually begin must be determined by calculating the TAB value MOD the width of the output device. In our example, the TAB value is 100, and the output device width is 40. The column where printing will begin can be calculated as 100 MOD 40, or the 20th column.

Two more points should be noted regarding TAB. First, TAB can be used with PRINT, but not with PRINT USING. Secondly, if the column being TAB'ed to is already occupied, the output will be generated at the same column on the next available line.

SPC

SPC sends a given number of spaces to the output device. The use of SPC is demonstrated in the following program:

Notice that the comma, which is a character with special meaning for numeric formatting, has no special meaning when used with strings. In other words, characters with special meaning for numeric formatting are literals when used for character formatting.

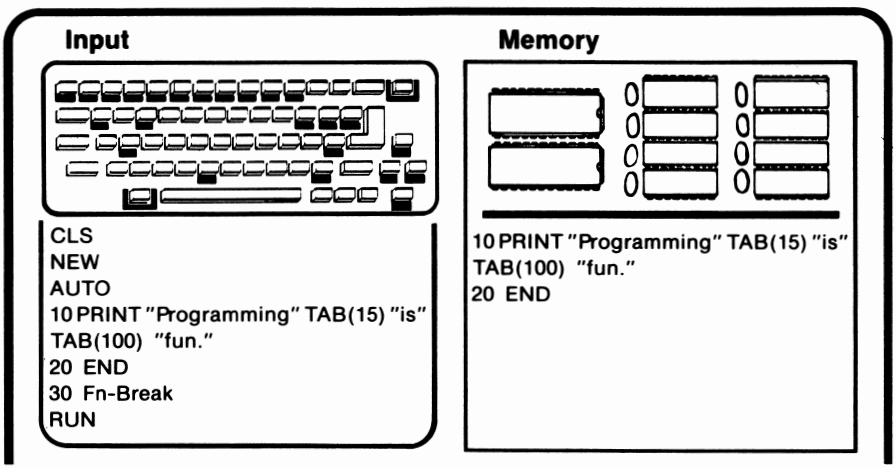
A formatting character can be treated as a literal by preceding the character with a _. When the line is executed, that character will be considered a literal. The underline will not appear.

Formatting Functions: TAB, SPC, SPACE$

PRINT USING is not the only option available in Microsoft BASIC for formatting output. Output can also be formatted by using TAB, SPC, and SPACE$.

TAB

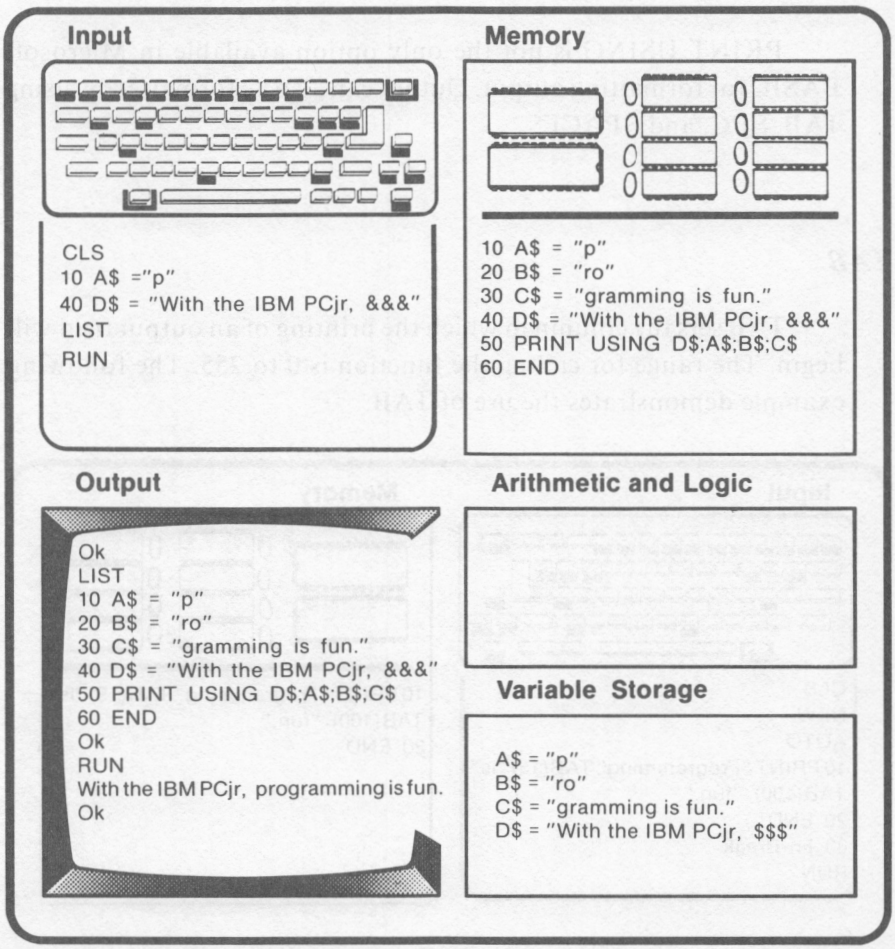
TAB sets the column in which the printing of an output field will begin. The range for calling the function is 0 to 255. The following example demonstrates the use of TAB:



! is most commonly used in creating tables, but ! can be helpful whenever a one character field needs to be output.

Literals

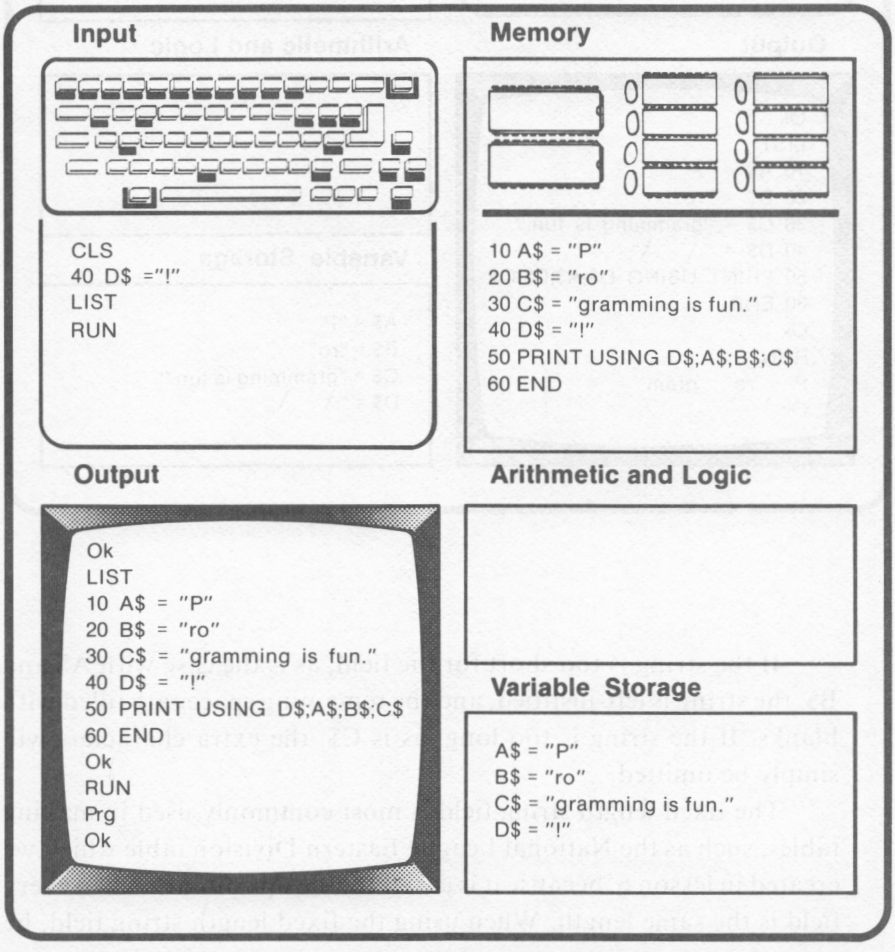
Literals are characters which have no special meaning as formatting characters (ex. 7, A, Z, @). When a literal character is included in a format string, it will be included in the output field exactly as it is specified in the format string.

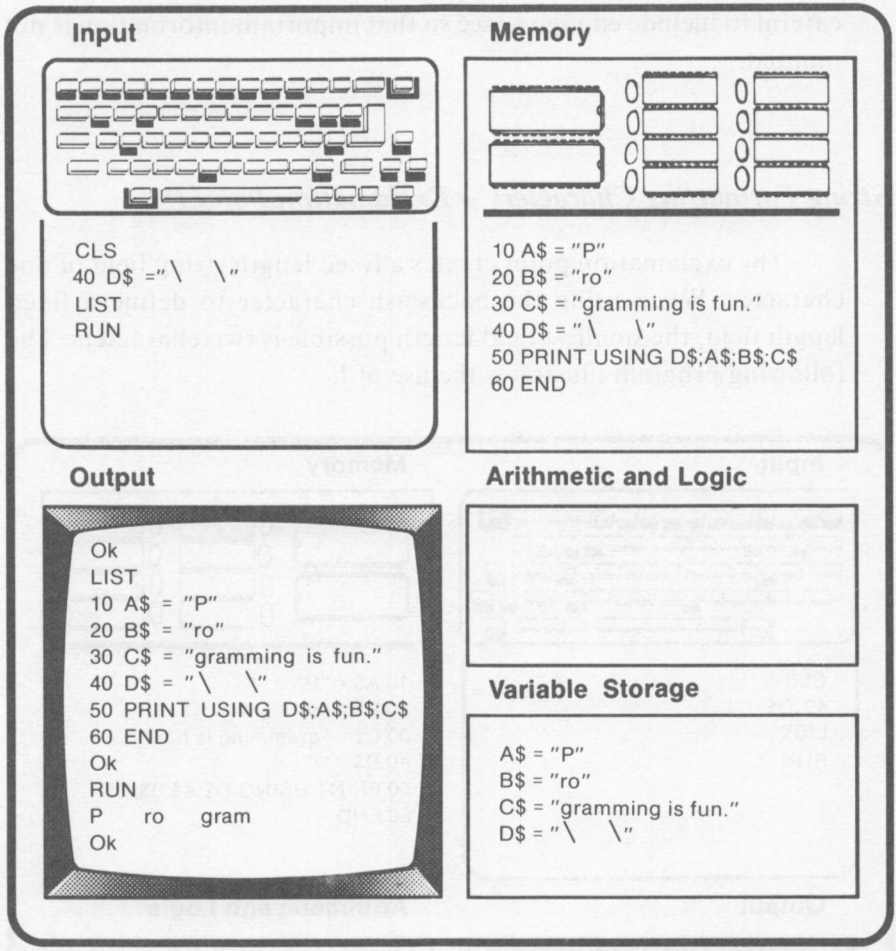


careful to include enough space so that important information is not omitted.

String Formatting Characters -- Exclamation Point (!)

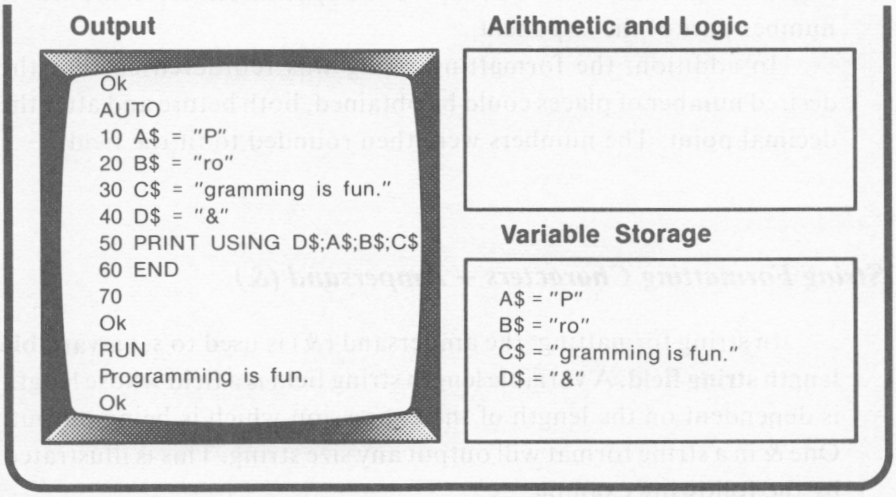
The exclamation point creates a fixed length string field of one character. When using the backslash character to define a fixed length field, the smallest field length possible is two characters. The following program illustrates the use of !:





If the string is too short for the field, as is the case with A$ and B$, the string is left-justified, and the remaining spaces are filled with blanks. If the string is too long, as is C$, the extra characters will simply be omitted.

The fixed length string field is most commonly used in making tables, such as the National League Eastern Division table which we created in lesson 6, because it is easier to line up columns when every field is the same length. When using the fixed length string field, be



String Formatting Characters -- Backslash (\ )

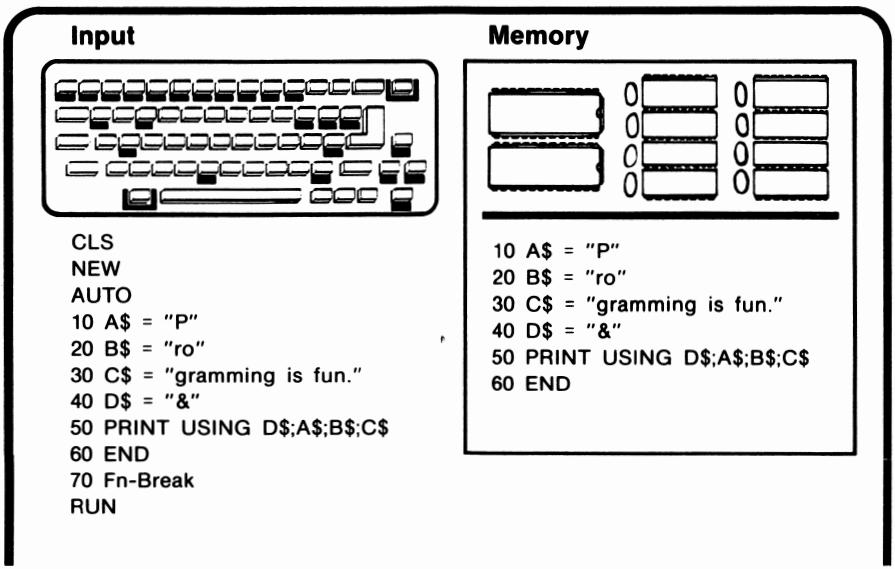
Two backslashes (\ \) define a fixed length string field. The number of characters which can be held in each field is dependent upon the number of spaces the backslashes enclose plus two. The extra two characters are reserved by the two backslash characters. The backslash is the same character as the integer division character. The use of the backslash as a formatting character is shown in the following example:

negative sign would have occupied the spot allotted for the whole number part of the exponent.

In addition, the formatting string was reordered so that the desired number of places could be obtained, both before and after the decimal point. The numbers were then rounded to fit the field.

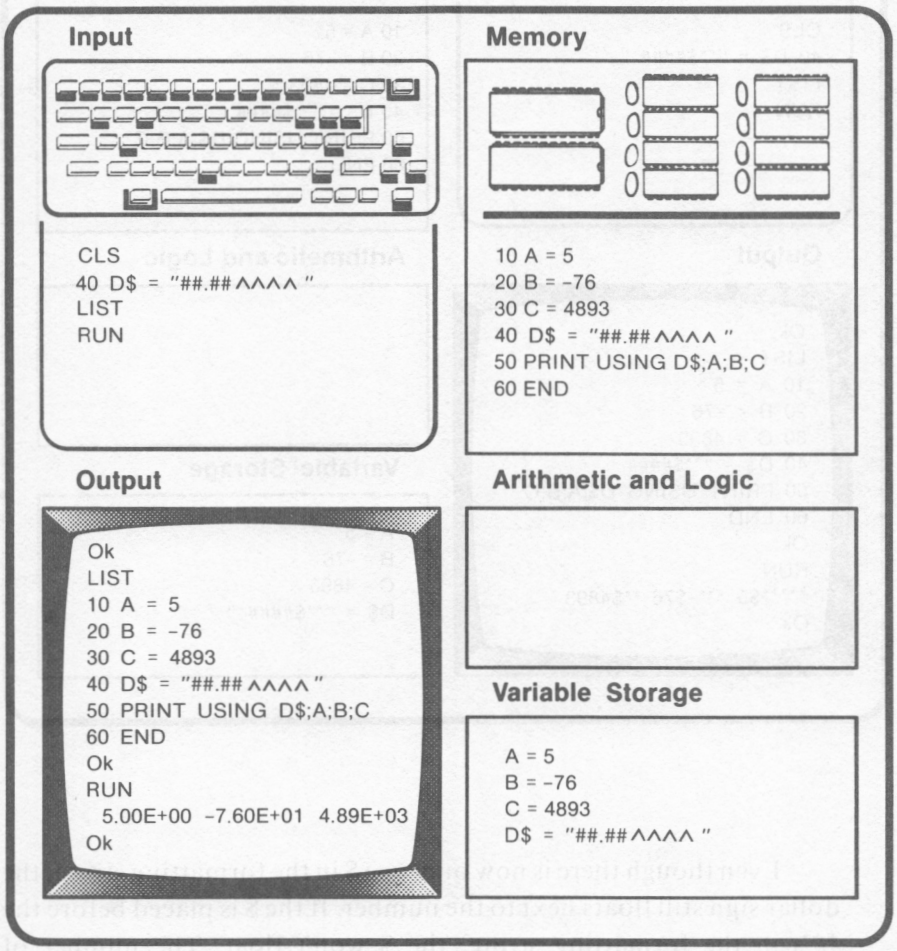
String Formatting Characters -- Ampersand (&)

In string formatting, the ampersand (&) is used to set a **variable length string field.** A variable length string field is a field whose length is dependent on the length of the expression which is being output. One & in a string format will output any size string. This is illustrated by the following example:

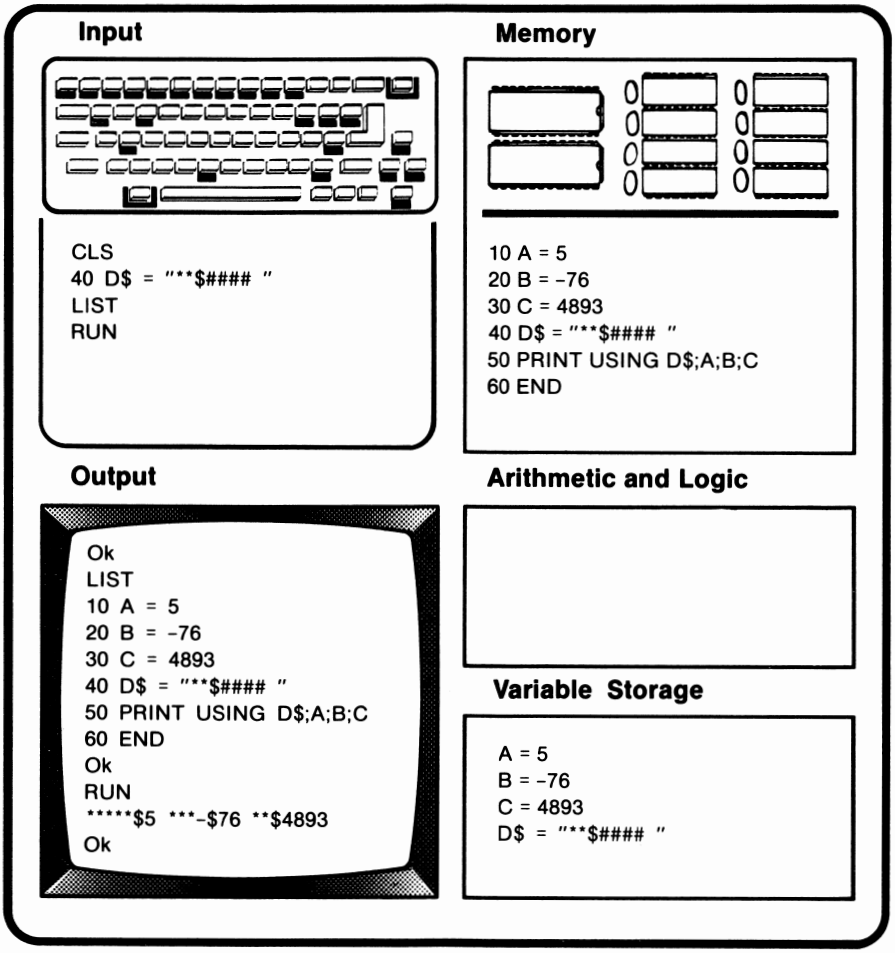


Numeric Formatting Characters -- Exponential Notation (^^^^)

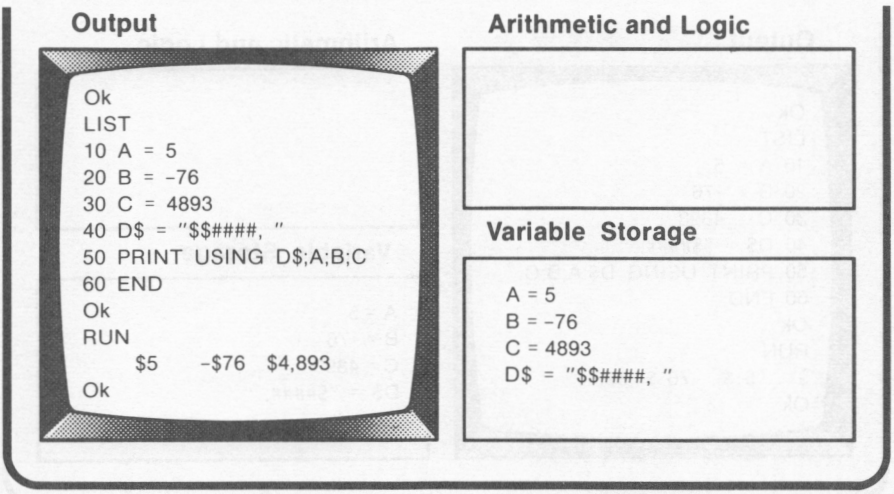
The use of 4 carets after the decimal point in the formatting string causes the number to be written in exponential notation. Let's examine an example:



A number of changes occurred in this example. An additional # was placed prior to the decimal point so that the negative sign would have room to appear. If that additional # had been omitted, the



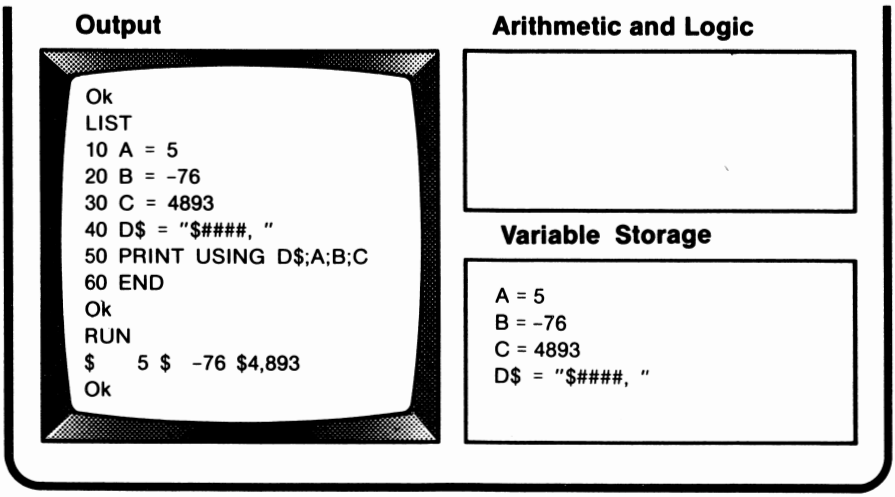
Even though there is now only one $ in the formatting string, the dollar sign still floats next to the number. If the $ is placed before the **'s in the formatting string, the $ won't float. The number of available spaces in each output field has again increased, since each * reserves a space.



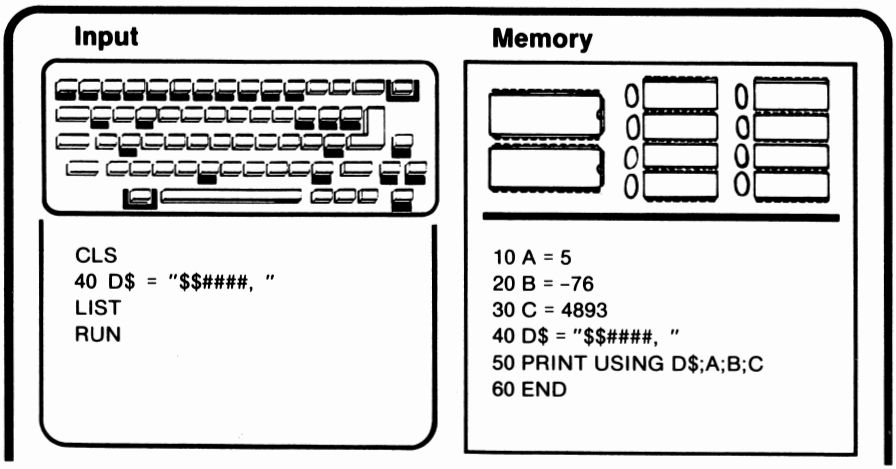
Notice that the dollar sign appears after the negative sign in the output. Also, the additional $ allots one more space in the field. If this space is not needed, it will be filled with a blank space.

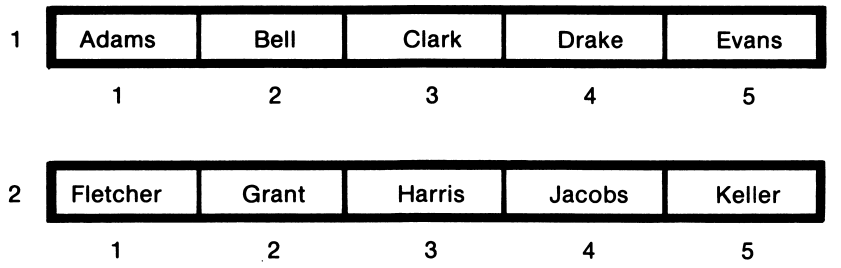
Numeric Formatting Characters -- Asterisk (*)

If two asterisks (**) are placed in the leftmost positions of the formatting string, the blank spaces will be replaced by *'s. This option is used most often when checks are being written, because it increases the difficulty of altering the amounts. This point is demonstrated by the following example:



When the single $ was used, the $ was printed in the leftmost position of each field. The following example demonstrates the use of two dollar signs:





Despite the added dimension, each subscripted variable is still unique. It is not necessary to limit our table to just 5 rooms or just 2 floors. The limits to table size depend only upon the space available in memory.

Notice that in this example we used nested FOR, NEXT loops. FOR, NEXT loops are often used when working with subscripted variables. In this program, the outer FOR loop signified the first dimension, the floor. The floor can also be referred to as the **row,** meaning the horizontal dimension. The row is the first number in the subscript. All of the residents of the first floor are in row 1, and all of the residents of the second floor are in row 2.

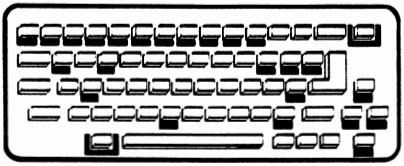
Both Adams and Fletcher have the same room number, 1. The room number is also known as the column. Columns, in tables or in architecture, run vertically. Adams and Fletcher are in the first column. Drake and Jacobs are in the fourth column.

DIM

When the previous program was executed, the maximum value of each dimension was set to 10 by default. Since the starting value, or **base,** of each dimension is assumed to be 0, an array can hold eleven elements. A table can hold 11 x 11, or 121, elements.

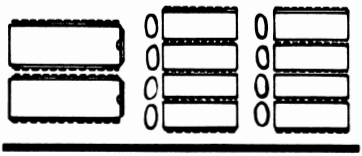
If 16 elements are needed in the array, we can tell the computer to set aside 16 spaces by using BASIC's DIM statement. DIM is short for dimension. DIM is used to define an array's dimensions.

Input



```
CLS
NEW
AUTO
10 DIM A(15)
20 FOR LOOP1 = 0 TO 15
30 A(LOOP1) = LOOP1 + 1
40 NEXT LOOP1
50 FOR LOOP2 = 0 TO 15
60 PRINT A(LOOP2);
70 NEXT LOOP2
80 END
90 Fn-Break
RUN
```

Memory



```
10 DIM A(15)
20 FOR LOOP1 = 0 TO 15
30 A(LOOP1) = LOOP1 + 1
40 NEXT LOOP1
50 FOR LOOP2 = 0 TO 15
60 PRINT A(LOOP2);
70 NEXT LOOP2
80 END
```

Arithmetic and Logic

```
 0 + 1 = 1      0 + 1 = 1      1 + 1 = 2
 1 + 1 = 2      2 + 1 = 3      2 + 1 = 3
 3 + 1 = 4      3 + 1 = 4      4 + 1 = 5
 4 + 1 = 5      5 + 1 = 6      5 + 1 = 6
 6 + 1 = 7      6 + 1 = 7      7 + 1 = 8
 7 + 1 = 8      8 + 1 = 9      8 + 1 = 9
 9 + 1 = 10     9 + 1 = 10    10 + 1 = 11
10 + 1 = 11    11 + 1 = 12    11 + 1 = 12
12 + 1 = 13    12 + 1 = 13    13 + 1 = 14
13 + 1 = 14    14 + 1 = 15    14 + 1 = 15
15 + 1 = 16    15 + 1 = 16
```

Output

```
Ok
NEW
Ok
AUTO
10 DIM A(15)
20 FOR LOOP1 = 0 TO 15
30 A(LOOP1) = LOOP1 + 1
40 NEXT LOOP1
50 FOR LOOP2 = 0 TO 15
60 PRINT A(LOOP2);
70 NEXT LOOP2
80 END
90
Ok
RUN
 1  2  3  4  5  6  7  8  9  10
  11  12  13  14  15  16
Ok
```

Variable Storage

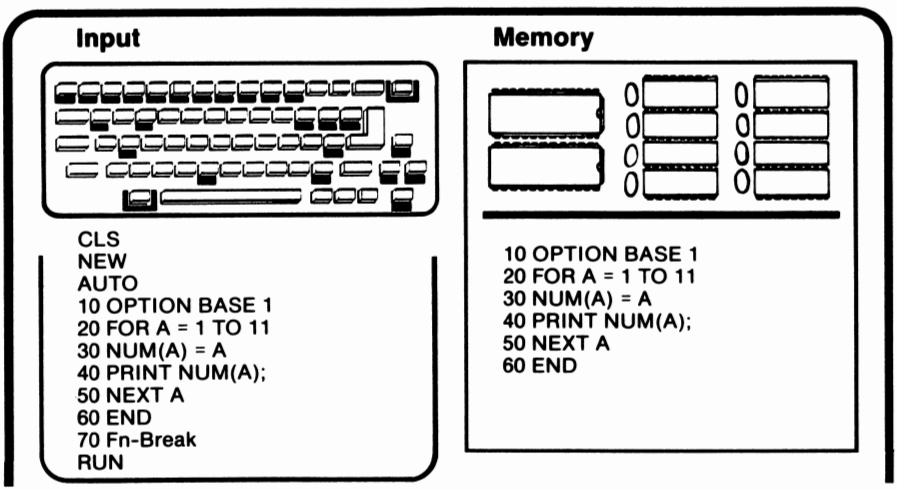
```
LOOP 1 = 0     A(0)       = 1
LOOP 1 = 1     A(1)       = 2
LOOP 1 = 2     A(2)       = 3
LOOP 1 = 3     A(3)       = 4
LOOP1 = 4      A(4)  = 5
LOOP1 = 5      A(5)  = 6
LOOP1 = 6      A(6)  = 7
LOOP1 = 7      A(7)  = 8
LOOP1 = 8      A(8)  = 9
LOOP1 = 9      A(9)  = 10
LOOP1 = 10     A(10) = 11
LOOP1 = 11     A(11) = 12
LOOP1 = 12     A(12) = 13
LOOP1 = 13     A(13) = 14
LOOP1 = 14     A(14) = 15
LOOP1 = 15     A(15) = 16
LOOP1 = 16
LOOP2 = 0      LOOP2 = 1      LOOP2 = 2
LOOP2 = 3      LOOP2 = 4      LOOP2 = 5
LOOP2 = 6      LOOP2 = 7      LOOP2 = 8
LOOP2 = 9      LOOP2 = 10     LOOP2 = 11
LOOP2 = 12     LOOP2 = 13     LOOP2 = 14
LOOP2 = 15     LOOP2 = 16
```

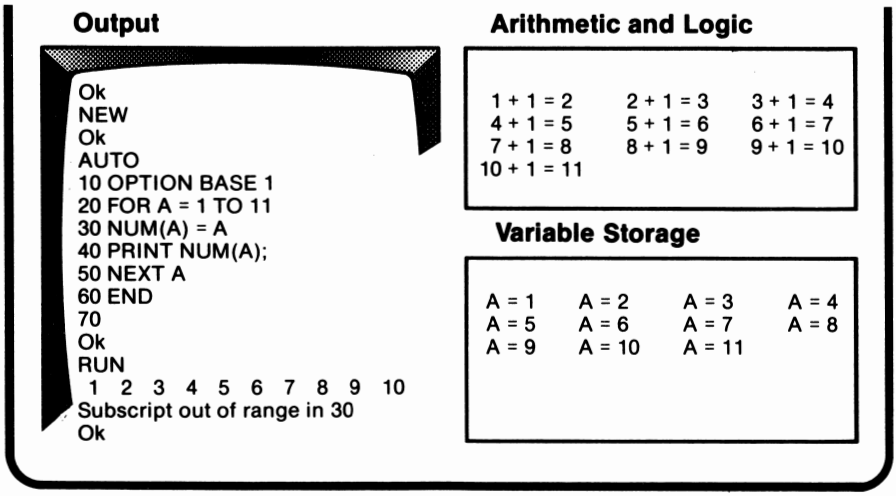
The memory space set aside for a table can also be increased by using DIM. DIM(20,20), for example, sets aside space in memory for a table with 441 elements. Two or more variables can be redimensioned in one statement by separating the variables with a comma. An example is DIM Q(50),B(25,30). DIM statements are usually grouped at the beginning of a program so that they are easy to find and so that arrays and tables are dimensioned before they are referenced in the program.

OPTION BASE

OPTION BASE is used to specify the beginning subscript for all array variables. By default, the base is 0. When the base is 0, DIM A(20) means that 21 spaces are available in memory for A. This can be a source of confusion as beginning programmers often interpret DIM A(20) as reserving 20 spaces in memory instead of 21.

When OPTION BASE is used, the most common value specified is 1. A value of 1 is used to avoid the confusion caused by allowing a subscript of zero. With OPTION BASE set to 1, DIM A(20) reserves 20 spaces in memory for A, not 21.

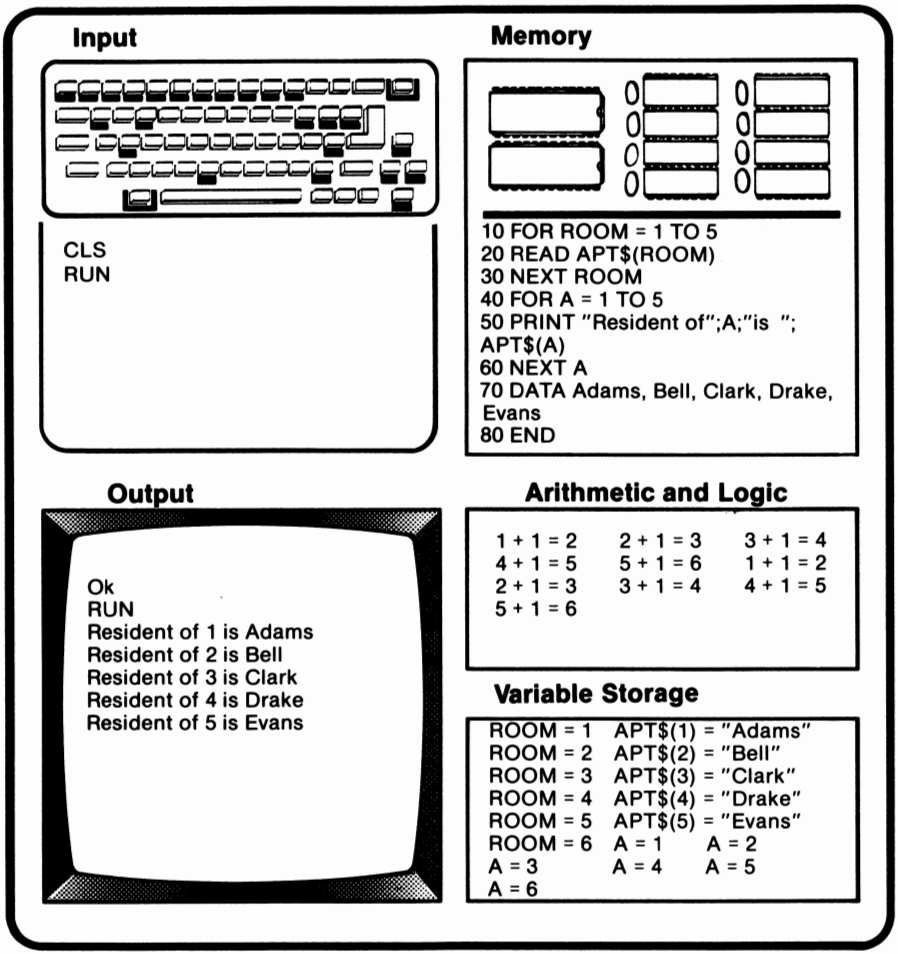




When we tried to use eleven spaces, we discovered that there were only 10 available. Zero is no longer a valid subscript. OPTION BASE is usually placed prior to any DIM statements.

DATA and READ

DATA lists the values to be assigned to variables. DATA is more efficient than INPUT when a large number of variables need data values assigned. READ assigns the data values specified in the DATA statement to the variables included with it. Let's examine an example of DATA and READ:



When using DATA, a string need only be enclosed in quotation marks if it contains a comma or a colon, or if it begins with a blank space. Reserved words can be used as data.

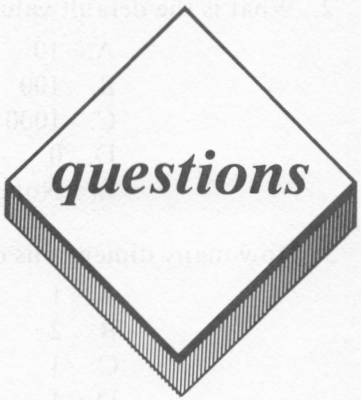
When READ is executed, the computer searches for DATA. The variables indicated with READ are assigned values from the DATA statement one by one. If the first DATA statement contains

fewer data items than corresponding READ variables, the computer will search for another DATA statement. The computer keeps track of which data item is to be assigned next with an internal pointer.

If there are more READ variables than DATA values, an Out of DATA error message will appear. A syntax error will occur if the READ variable and the DATA value don't agree in type.

ERASE

ERASE eliminates variable storage. When an array is no longer needed, ERASE frees the memory previously assigned to it, allowing the space to be reused. ERASE is more commonly used when working with large arrays. ERASE APT$ is an example of an ERASE command.



True or False

1. A(1) and A(2) use the same space in memory.
2. Columns run horizontally and are specified by the first number in a subscript. Rows are vertical and are specified by the second number.
3. Any action which can be performed on string variables can be performed on subscripted string variables.
4. The default value for OPTION BASE is 1.
5. In DATA statements, all strings must be enclosed in quotation marks.

Multiple Choice

1. What does ERASE do?

   A. Clears the screen
   B. Frees memory
   C. Sets all variables equal to 0
   D. Sets the lowest subscript to 0
   E. None of the above

2. What is the default value for DIM?

   A. 10
   B. 100
   C. 1000
   D. 0
   E. None of the above

3. How many dimensions does a table have?

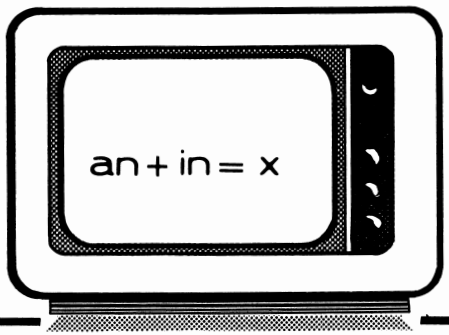
   A. 1
   B. 2
   C. 3
   D. 4
   E. None of the above

4. In A(3,21) what is the subscript?

   A. A
   B. 3
   C. 21
   D. 3,21
   E. None of the above

Computer Exercises

1. Revise the program you wrote for Computer Exercise 1 on page 323 to do the following:
   a. Use subscripted variables for the months.
   b. Use DATA and READ statements instead of INPUT.



Numeric and Math Functions

lesson 17

Lesson Goals

- *Learn how to use BASIC's trigonometric functions: SIN, COS, TAN, and ATN*
- *Learn how to use BASIC's SQR, INT, FIX, ABS, SGN, EXP, and LOG functions*
- *Learn how to use BASIC's CINT, CSNG, and CDBL functions to convert the types of numerics*

Introduction

In this lesson we will discuss numeric and math functions. A **function** defines a set of operations to be performed on a numeric or string value. The format for using a BASIC function is:

Function name (data value)

The function name is followed by a data value. This data value can be either a constant or a variable and is enclosed in parentheses. For consistency, the data value will be referred to as X.

The functions which we will be discussing in this lesson and in the next lesson are **built-in** functions. Built-in means that they are a part of the Microsoft BASIC interpreter. Using a function is known as **calling** a function. A synonym for calling is invoking. This process is referred to as calling because the definitions of the operations to be executed are located elsewhere. For built-in functions, that definition is part of the BASIC interpreter.

SIN, COS, TAN, ATN

SIN, COS, TAN, and ATN are trigonometric functions. Trigonometry is the branch of mathematics which deals with the relationship of the sides and angles of a triangle. In the following figure, X is the angle, a is the **opposite** side, b is the **adjacent** side, and c is the **hypotenuse,** or side opposite the right angle:

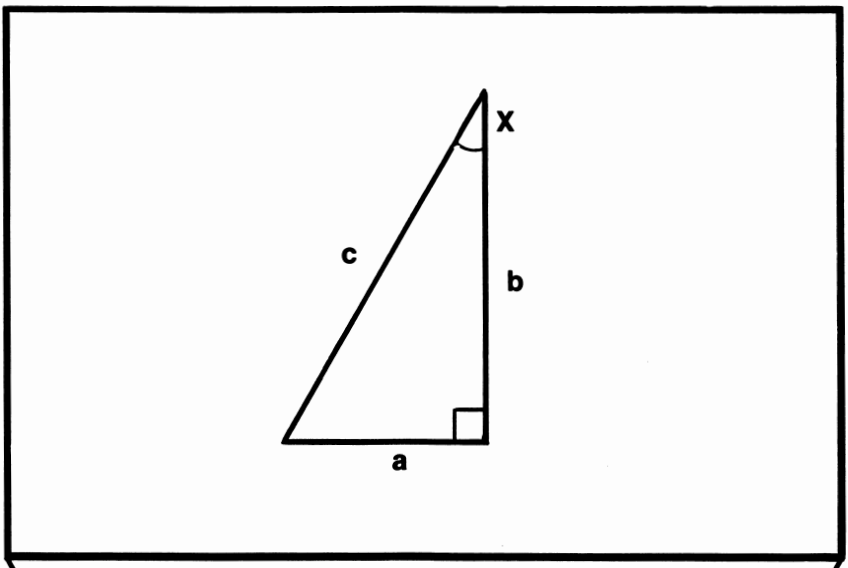


Figure 17.1. Example triangle

SIN(X) returns the sine of the angle X. The sine of X is the length of the side opposite the angle, a, divided by the length of the hypotenuse, c.

Calling COS(X) results in the cosine of X being returned. Cosine is defined as the length of the side adjacent to angle X, b, divided by the length of the hypotenuse, c.

TAN(X) returns the tangent of X. The tangent is the length of the side opposite the angle divided by the length of the side adjacent to the angle. In this case, the tangent of X is equal to the length of a divided by the length of b.

Calling ATN(X) returns the arctangent of X. The arctangent of X is the angle whose tangent is X. If TAN(X) is called and the value returned by that call is used to call ATN, X is returned.

For COS, SIN, and TAN, X must be specified in **radians.** A radian is equal to 57.29578 degrees. One degree equals .017453 radians. ATN returns the measure of the angle in radians. The following program illustrates the use of the trigonometric functions:

```
Ok
LIST
 10 RAD = .017453
 20 DEG = 57.29578
 30 FOR L = 1 TO 3
 40 READ NUM(L)
 50 S(L) = SIN(NUM(L) * RAD)
 60 C(L) = COS(NUM(L) * RAD)
 70 T(L) = TAN(NUM(L) * RAD)
 80 A(L) = ATN(T(L)) * DEG
 90 PRINT NUM(L);S(L);C(L);T(L);A(L)
100 NEXT L
110 DATA 30, 45, 60
120 END
Ok
RUN
  30   .4999924   .8660299   .5773385
 29.9995
  45   .7070975   .7071161   .9999737
 44.99925
  60   .8660166   .5000153   1.73198
 59.999
Ok
```

In this example the values for X are given in degrees. Prior to calling SIN, COS, and TAN, the values are converted to radians. The SIN, COS, and TAN functions are then called and the results are assigned to subscripted variables. The value assigned to T(L) was returned by TAN. This value is used to call ATN. The result of ATN is multiplied by the number of degrees which equal one radian. Notice that this final value is approximately equal to X. The data is then output.

SQR

SQR is the built-in square root function. The square root of a number is the number which, if squared, will result in the original

number. The square root of 81, for example, is 9, since nine squared equals 81. The following program demonstrates the use of SQR:

```
Ok
LIST
10 FOR L = 1 TO 3
20 READ NUM(L)
30 S(L) = SQR(NUM(L))
40 PRINT NUM(L);S(L);S(L) * S(L)
50 NEXT L
60 DATA 25, 30, 50
70 END
Ok
RUN
 25   5   25
 30   5.477226   30
 50   7.071068   50
Ok
```

During execution of this program, the data value is read, and its square root is computed and assigned to the subscripted variable, S. Execution of line 40 causes the number, its square root, and its square root times itself to be output. Notice that multiplying the number stored in S times itself returns the original value.

INT

BASIC's INT function returns the integer value of its argument, or calling value. INT returns the highest integer whose value is less than or equal to the argument's value. If X is a positive number, INT(X) will return the integer portion of X. INT(3.691), for example, would return 3. If X is negative, INT(X) returns the next lower integer. An example of this point is that INT(-6.10938) would return -7. The INT function is demonstrated in the following example:

```
Ok
LIST
10 FOR L = 1 TO 3
20 READ NUM(L)
30 I(L) = INT(NUM(L))
40 PRINT NUM(L);I(L)
50 NEXT L
60 DATA 2.6, 1.1, -8.3
70 END
Ok
RUN
 2.6   2
 1.1   1
-8.3  -9
Ok
```

INT(2.6) returns 2 because 2 is the highest integer less than or equal to 2.6. One is the highest integer whose value is less than or equal to 1.1, so it is returned. -8 is greater than -8.3, so -9 is returned instead of -8 when INT is called.

FIX

FIX(X) and INT(X) have the same effect when X is positive or 0. Since FIX simply discards the decimal portion, FIX(3.691) returns 3, and FIX(-6.10938) returns -6, not -7. The use of FIX can be seen in the following program:

```
Ok
LIST
10 FOR L = 1 TO 3
20 READ NUM(L)
30 F(L) = FIX(NUM(L))
40 PRINT NUM(L);F(L)
50 NEXT L
60 DATA 2.6, 1.1, -8.3
70 END
Ok
RUN
 2.6  2
 1.1  1
-8.3 -8
Ok
```

While INT(-8.3) returned -9, FIX(-8.3) returns -8. FIX discards the decimal portion. It is not concerned with whether the value returned is greater or less than the argument.

ABS

ABS returns the absolute value of its argument. Absolute value is the distance of that number from 0. If X is zero or positive, the value returned equals X. This result is due to the fact that the distance from the value to 0 is equal to that value. When working with negative numbers, the absolute value of the number is that number without the negative sign. ABS never returns a negative value, because distance is positive. -3 and 3 are both 3 units from zero.

```
Ok
LIST
10 FOR L = 1 TO 3
20 READ NUM(L)
30 A(L) = ABS(NUM(L))
40 PRINT NUM(L);A(L)
50 NEXT L
60 DATA 2, 0, -8
70 END
Ok
RUN
 2  2
 0  0
-8  8
Ok
```

ABS(2) equals 2, since by definition, 2 is two units away from 0. ABS(0) equals 0. ABS(–8) returns 8, because –8 is 8 units away from zero.

SGN

SGN returns a value which indicates the sign of its numeric argument. If X is positive, SGN(X) returns a value of 1. If X is negative, –1 is returned. If X equals 0, 0 is returned.

```
Ok
LIST
10 FOR L = 1 TO 3
20 READ NUM(L)
30 S(L) = SGN(NUM(L))
40 PRINT NUM(L);S(L)
50 NEXT L
60 DATA 2, 0, -8
70 END
Ok
RUN
 2  1
 0  0
-8 -1
Ok
```

Since two is a positive number, 1 is used to indicate its sign. Zero is neither positive nor negative, so a value of 0 is returned. Finally, -8 is a negative number, its sign is represented by a value of -1.

EXP

EXP(X) returns the value of the base raised to a specified value. The specified value is the calling argument, X. When working with EXP, the base is assumed to equal 2.71828183. This value is also known as e. The following equation represents EXP:

$$y = e^x$$

X is the exponent as well as the calling argument. e is the base, and y is the value returned by EXP(X). The calling value cannot be greater than 88.02969, because the computer is not designed to

handle extremely large numbers. Using a larger number will cause an overflow.

```
Ok
LIST
10 FOR L = 1 TO 3
20 READ NUM(L)
30 E(L) = EXP(NUM(L))
40 PRINT NUM(L);E(L)
50 NEXT L
60 DATA 2.772589, 0, 4.158883
70 END
Ok
RUN
 2.772589   16.00001
 0   1
 4.158883   64
Ok
```

In this example, e was raised to each of the data values. 2.71828183 ^ 2.2589 equals approximately 16. e times itself 0 times is equal to 1. Finally, 3 to the 4.158883 power equals 64.

LOG

LOG uses the value of y as its calling argument and returns X. This is also known as the natural logarithm of X. A logarithm is the exponent of the power to which a base number must be raised to equal a given number. LOG, by definition, cannot be called by a negative number. Using LOG on the answers we generated in the last example will return the original value of X:

```
Ok
LIST
10 FOR L = 1 TO 3
20 READ NUM(L)
30 LG(L) = LOG(NUM(L))
40 PRINT NUM(L);LG(L)
50 NEXT L
60 DATA 16, 1, 64
70 END
Ok
RUN
  16   2.772589
  1   0
  64   4.158883
Ok
```

LOG(16) returned 2.772589. LOG(1) returned 0, and LOG(64) returned 4.158883.

CINT, CSNG, CDBL

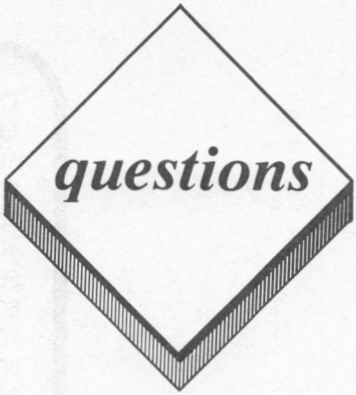
These functions convert numbers to a specified type. CINT(X) converts X to an integer, and CSNG(X) changes X to a single precision value. CDBL(X) results in X being converted to double precision. If necessary, the values are rounded. The following program illustrates the use of these functions:

```
Ok
LIST
10 FOR L = 1 TO 3
20 READ NUM#(L)
30 CI(L) = CINT(NUM#(L))
40 CS(L) = CSNG(NUM#(L))
50 CD#(L) = CDBL(NUM#(L))
60 PRINT NUM#(L);CI(L);CS(L);CD#(L)
70 NEXT L
80 DATA 1.356, 8, 67.987654321
90 END
Ok
RUN
 1.356   1   1.356   1.356
 8   8   8   8
 67.987654321   68   67.98766   67.987654321
Ok
```

CI(1.356) in integer form is merely 1. No rounding is needed. Since the number is already single precision, it is not altered by CSNG. In double precision, 1.356 is represented as 1.356, also.

When the integer, 8, is used as X, it is output as 8, regardless of its type.

Finally, 67.987654321 is rounded to 68 in order to be stored as an integer. In single precision, its value is 67.98766, rounded to seven digits. Since the number is already double precision, CDBL has no effect.



True or False

1. Microsoft BASIC's COS built-in function is used to find an angle's cosecant.
2. The SQR function returns the value of its argument squared.
3. INT(X) and FIX(X) always return the same value.
4. ABS returns the sign of the number.
5. A function defines the operations to be performed on string or numeric values.

Multiple Choice

1. Which of the following functions returns the measure of an angle?

   A. SIN
   B. COS
   C. ATN
   D. TAN
   E. None of the above

2. Which of the following functions returns the distance of a number from 0?

   A. ABS
   B. SGN
   C. INT
   D. FIX
   E. None of the above

3. Which of the following functions will convert a number to single precision?

   A. INT
   B. FIX
   C. SQR
   D. CSNG
   E. None of the above

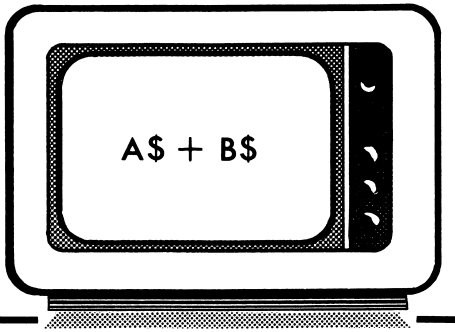
4. What is the maximum allowed value with which to call EXP?

   A. e
   B. 2.71828183
   C. 32647
   D. 88.02969
   E. None of the above

Computer Exercises

1. Use the computer to evaluate the following expressions:

   a. SIN(15) COS(15) TAN(15)
   b. SQR(15)
   c. INT(-2.38) FIX(-2.38)
   d. ABS(-2.38) SGN(-2.38)
   e. EXP(0) LOG(1)
   f. CINT(3.63) CSNG(3.63) CDBL(3.63)



String Functions

lesson 18

Lesson Goals

- *Learn how to concatenate strings*
- *Learn how to use BASIC's LEFT$, RIGHT$, and MID$ functions*
- *Learn how to use BASIC's STR$, VAL, CHR$, and ASC functions*
- *Learn how to use BASIC's INSTR function*
- *Learn how to find the length of a string by using BASIC's LEN function*
- *Learn how to use BASIC's STRING$ function*

Introduction

In this lesson we will work with BASIC's built-in functions which apply to strings. These functions will permit us to add strings together, select specified characters for use, convert data types, and search for selected characters.

String Concatenation

Two strings can be added together to create a new string. This merging is called **concatenation.** The following program demonstrates this process:

```
Ok
LIST
10 A$ = "Programming "
20 B$ = "is fun."
30 C$ = A$ + B$
40 PRINT C$
50 END
Ok
RUN
Programming is fun.
Ok
```

In this example two strings, A$ and B$, were merged into one string, C$. A$ and B$ were not affected by the concatenation. The symbol for string concatenation is the plus sign, +.

LEFT$

LEFT$ returns a specified number of characters beginning with the leftmost character. The configuration for the function call is:

LEFT$(string, number of characters desired)

LEFT$ requires two parameters. The first argument, the string, is the string from which the characters are to be selected. The second parameter is the number of characters to be selected. If the second parameter is greater than the total number of characters within the string, the entire string will be returned. The following program illustrates the operation of LEFT$:

```
Ok
LIST
10 A$ = "Programming is fun."
20 B$ = LEFT$(A$,5)
30 PRINT B$
40 END
Ok
RUN
Progr
Ok
```

During execution of line 20, the five leftmost characters are taken and assigned to B$. Again, A$ is not affected by the function call.

RIGHT$

RIGHT$ also returns a specified number of characters from a string. The key distinction between LEFT$ and RIGHT$ is that RIGHT$ begins with the rightmost characters of the string, not the leftmost. The format for calling RIGHT$ is similar to the format for calling LEFT$:

RIGHT$(string, number of characters selected)

The two required arguments are the initial string and the number of desired characters. The following program demonstrates the operation of RIGHT$:

```
Ok
LIST
10 A$ = "Programming is fun."
20 B$ = RIGHT$(A$,5)
30 PRINT B$
40 END
Ok
RUN
 fun.
Ok
```

In this example the 5 characters at the end of the string were assigned to B$. A$ was not affected. It is important to note that the characters are not reordered by using RIGHT$.

MID$

MID$ is used to select characters in the middle of a string. When working with LEFT$ and RIGHT$, selection must begin at one of the ends of the string. With MID$, selection can begin at any point within the string. MID$'s format is as follows:

MID$(string, starting character's position, number of characters)

The string and the starting position are required. The starting positon is a number which indicates the position of the specified character within the string. In the string "Hello", for example, "H" is in position 1, and "o" is located in position 5.

The third parameter, the number of characters, is optional. If the number of characters to be selected is not specified, the remainder of the string, beginning at the specified starting position, is returned. The following program portrays the use of MID$:

```
Ok
LIST
10 A$ = "Programming is fun."
20 B$ = MID$(A$,5,6)
30 PRINT B$
40 END
Ok
RUN
rammin
Ok
```

Notice that six characters were returned and assigned to B$, beginning with the "r" in position 5. The characters were not reordered, and A$ was not affected.

MID$ can also be used to replace characters in one string with characters from another string. If the replacement string has fewer characters than the value of the third parameter, the substitution will begin with the character in the initial string which is in the specified starting position. Additional characters will simply be retained. In the string "Hello", for example, if we wanted to replace the characters "ello" with "bye" the result would be "Hbyeo". The string "bye" has fewer characters, 3, than does the string "ello", so the "o" is retained. The following program illustrates the use of MID$ to replace characters:

```
Ok
LIST
10 A$ = "Programming is fun."
20 MID$(A$,5,6) = "hello!"
30 PRINT A$
40 END
Ok
RUN
Proghello!g is fun.
Ok
```

In this case, A$ was altered. Note that the replacement string is assigned to the specified area in the initial string. It is not necessary to assign that string to A$, as that assignment occurs automatically.

STR$ and VAL

The BASIC functions of STR$ and VAL perform string-numeric conversion. STR$ converts a numeric value into a string. VAL converts a string into a numeric value. The string to be con-

verted by VAL must consist of numeric characters such as "91". If a blank space appears in the string, it will be ignored. If any other nonnumeric character appears, the numeric characters up to the nonnumeric will be returned. The functions have the following formats:

STR$(numeric value)
VAL(string)

The following program illustrates the operation of these functions:

```
Ok
LIST
10 A$ = STR$(1984)
20 B = VAL("1984")
30 PRINT A$,B
40 END
Ok
RUN
 1984            1984
Ok
```

Execution of line 10 causes the value 1984 to be converted to a string. That string is then assigned to A$. Execution of line 20 causes the string, "1984", to be converted to its numeric value of 1984 and assigned to B.

CHR$ and ASC

CHR$ and ASC perform ASCII-text character conversions. These functions only operate on one ASCII code or one text character at a time. CHR$ converts an ASCII code value, such as 83, into a text character. The ASC function converts a text character into an ASCII code value.

The functions' formats are:

CHR$(ASCII code)
ASC(text character)

The text character used to call ASC must be enclosed in quotation marks since it is a string. The following program displays the use of CHR$ and ASC:

```
Ok
LIST
10 A$ = CHR$(100)
20 B = ASC("Q")
30 PRINT A$,B
40 END
Ok
RUN
d                81
Ok
```

The character with the ASCII code of 100 is "d". Execution of line 30 causes this character to be output. Since the ASCII code for "Q" is 81, that value is also output during line 30's execution.

INSTR

INSTR searches for the initial appearance of a specified string within another string. Once that appearance is found, INSTR returns the position where the match begins. The format for INSTR is:

INSTR(starting position, string to be searched, string to search for)

The latter two parameters are required. The first of these required parameters is the string to be searched. The second parameter is the character or characters for which we are searching. If, for example, we want to know if the letter "y" appears in the string "goodbye", "goodbye" is the string to be searched and "y" is the string for which to search.

The first parameter, the starting position, is optional. This parameter specifies the position where the search is to begin. If we want to know only if "y" occurs in the last half of the word, "goodbye", the starting position would be specified as 4. If we want the search to include the entire string, we can omit this parameter.

Capital and small letters are considered to be different characters. If we ask, instead, whether the letter "Y" appears in "goodbye", we will be told that it does not appear. If the string to be searched for is not found, a value of 0 will be returned.

The following program demonstrates the operation of INSTR:

```
Ok
LIST
10 A$ = "Programming is fun."
20 B = INSTR(A$,"i")
30 PRINT B
40 END
Ok
RUN
  9
Ok
```

Since no starting position was specified, the search began with "P". If a position is specified and the string is found, the value for the position will be counted from the beginning of the string to be searched, not from the beginning of the searched part of that string. If the search in our example had begun with the fourth character instead of the first, the value returned would still have been 9, since "i" appears for the first time in position 9. After the string was found, the search was halted. INSTR returns only the initial position.

LEN

LEN returns the number of characters in a string. In other words, LEN returns the length of the string. The format for calling LEN is:

LEN(string)

The string can be either a constant or a variable, as can the strings used by all of the functions we have discussed in this lesson. The following program demonstrates the use of LEN:

```
Ok
LIST
10 A$ = "Programming is fun."
20 B = LEN(A$)
30 PRINT B
40 END
Ok
RUN
 19
Ok
```

A$ has 19 characters, including punctuation and blank spaces. All characters are counted by LEN, and the value assigned to B is 19. Execution of line 30 causes this value to be output.

STRING$

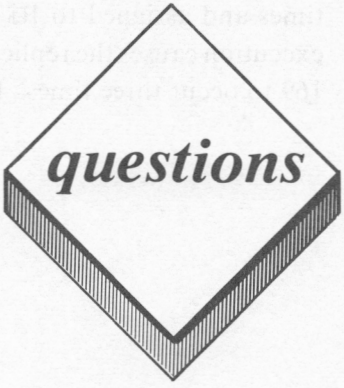
STRING$ returns a string of a specified length consisting of specified characters. The format for calling STRING$ is:

STRING$(number of characters desired, character desired)

The number of characters desired is the length of the string which is to be created. The character desired can be either a string or an ASCII code value. The string containing the desired character can consist of a number of different characters but only the first character will be replicated. This point can be more clearly seen in the following example:

```
Ok
LIST
10 A$ = "Programming is fun."
20 B$ = STRING$(5,A$)
30 C$ = STRING$(3,169)
40 PRINT B$,C$
50 END
Ok
RUN
PPPPP           ┌┌┌
Ok
```

The first character in A$ is "P". That character is replicated five times and assigned to B$ during the execution of line 20. Line 30's execution causes the replication of the character with the ASCII code 169 to occur three times. These strings are then output in line 40.



True or False

1. When two strings are concatenated to form a new string, the initial strings are not affected.
2. LEFT$ returns a specified number of characters from a string, beginning at its left-hand side.
3. MID$ can be used to replace characters in a string with characters from another string.
4. BASIC's VAL function is used to convert a numeric value to a string value.
5. LEN returns the number of characters in a string.

Multiple Choice

1. Which of the following symbols is used to concatenate strings?

   A. +
   B. ;
   C. ,
   D. –
   E. None of the above

2. What will be the output of the statement,
   MID$("Hello there !",2,4)?
   A. lo
   B. ello
   C. He
   D. ther
   E. None of the above

3. Which of the following functions can be used to convert a numeric value to a string?
   A. STR$
   B. VAL
   C. CHR$
   D. ASC
   E. None of the above

4. What will be the output of the statement,
   PRINT INSTR("Hello","l")?
   A. 6
   B. 3
   C. 0
   D. 3 4
   E. None of the above

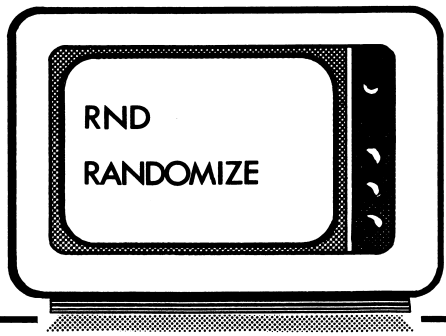
5. What will be the output of the statement,
   PRINT STRING$(2,"Hello")?
   A. He
   B. HH
   C. ll
   D. 2
   E. None of the above

Computer Exercises

1. Write a program to do the following:

   a. Input the user's first and last names as two strings.

b. Concatenate the strings and assign this new string to a variable.
c. Take the first 3 characters, the middle 3 characters, and the last 3 characters of the concatenated name. Assign these strings to variables.
d. Find the ASCII value for the 5th letter. Assign it to a variable.
e. Output the values for all the variables.



Other Functions and User-Defined Functions

lesson 19

Lesson Goals

- *Learn how to use BASIC's FRE function*
- *Learn how to use BASIC's POS function*
- *Learn how to use PEEK and POKE when working with memory*
- *Learn how to use RND and RANDOMIZE to generate numbers*
- *Learn the uses of BASIC's SCREEN function*
- *Learn how to create and utilize user-defined functions*

Introduction

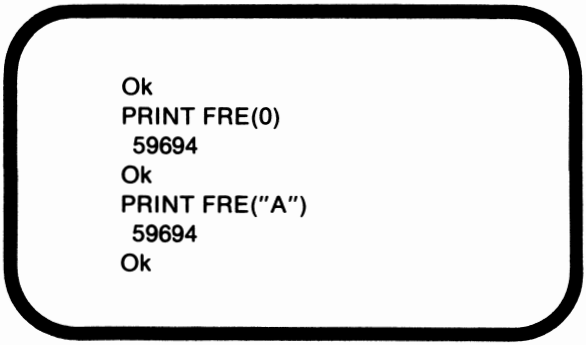
In the past two lessons, we have explored numeric and string functions. In this lesson we will discuss an assortment of built-in functions which cannot be assigned to either of these categories. We will also examine **user-defined functions.** User-defined functions are not built-in but are instead defined by the programmer.

FRE

FRE returns the number of unused bytes in memory. FRE can be called with either a string argument or a numeric argument. When FRE is used with a numeric argument, the function simply returns the number of unused bytes. If FRE is called using a string argument, **housekeeping** is performed prior to the return of the number of available bytes. Housekeeping consists of two steps. First, the useful data is gathered and stored in the smallest possible amount of space. This process releases the areas in memory which once held data but no longer do. Because data was once stored in these memory locations, that space is not considered available. Housekeeping frees this space. Then the number of available bytes is returned. FRE's format is the following:

FRE(numeric or string argument)

The following example illustrates the use of FRE:



In this example housekeeping had no effect. Housekeeping will be ineffectual whenever all memory which is in use is needed. If housekeeping is possible, the amount of memory available will increase after the process has been performed.

Under certain conditions housekeeping may be automatically performed. Housekeeping will automatically occur whenever the amount of space available in memory is less than that needed. The process will be performed in an attempt to release a sufficient number of bytes.

It is a good programming practice to use FRE with a string argument whenever a program is extremely long or when the tables and arrays which are being used are large. This process will determine whether enough space is available in memory for your program. If an insufficient number is available, the housekeeping will attempt to make enough bytes available.

POS

The column in which the cursor is currently located is returned by POS. Any string or numeric argument can be used to call POS. The format for calling POS is:

POS(string or numeric argument)

POS returns the same result regardless of whether a string argument or numeric argument is used as its argument. The following program demonstrates the use of POS:

```
Ok
LIST
10 FOR A = 1 TO 20
20 PRINT POS(0);
30 NEXT A
40 END
Ok
RUN
 1  4  7  10  14  18  22  26  30  34
 38  5  8  11  15  19  23  27  31  35
Ok
```

When POS is called for the first time, the cursor is located in the first column, so a value of 1 is returned. The cursor is then located in the fourth column. The second call of POS returns a value of 4. Notice that POS indicates only columns. It does not indicate the row. When POS is used and WIDTH is set to 40, the columns are numbered from 1 to 40. If WIDTH is 80, the columns are numbered from 1 to 80.

PEEK

PEEK returns the contents of a specified memory location. PEEK has the following format:

PEEK(memory location)

The specific memory location to be checked is represented by a whole number. The value for the memory location can range from 0 to 65535. PEEK returns an integer between 0 and 255, inclusive. The value returned is a representation of the contents of that memory location. A value of zero indicates that the space is empty. Since PEEK merely observes memory, it is most often used when one specific piece of information is needed. One possible use of PEEK, for example, is checking a location in memory to see if the button on a joystick has been pressed.

POKE

POKE actually alters the contents of a specified memory location. The configuration for POKE is:

POKE memory location, new contents

Notice that the arguments are not enclosed in parentheses. The first argument is a memory location. The second is the data which you want that location to contain. This data must be an integer within the range of 0 to 255, inclusive.

POKE actually changes the contents of a specific location in RAM memory. Because of its power, POKE is a risky command. As such, it is a good idea to use POKE sparingly and only when you are positive that you want to alter the contents of that specific location.

RND

RND is a function which generates a random number between 0 and 1. The number is considered to be random because every number between 0 and 1 has an equal chance of being selected. Flipping a coin

is an example of a random selection. Both heads and tails have an equal chance of being the result.

RND can be used with or without a parameter. The format for calling RND is:

RND(a)

The parameter, a, can be any numeric value. If a is positive, RND will return the next random number in the current sequence. An example of a possible sequence is .7151002, .683111, and .4821425. The first time RND is called with a positive value of a, the first number in the sequence is returned. The second call, using the example sequence, returns .683111. The third call returns .4821425.

Each time a program is executed, if RND'S argument is positive, the same sequence will be returned. To illustrate this point, execute the following program:

```
Ok
LIST
10 FOR X = 1 TO 5
20 PRINT RND(X)
30 NEXT X
40 END
Ok
RUN
 .7151002
 .683111
 .4821425
 .9992938
 .6465093
Ok
```

Suppose we then cleared the screen by executing CLS, edited line 10 as follows:

```
10 FOR X = 20 to 24
```

and ran the program a second time. As shown below, the same series of random numbers would be generated:

```
Ok
EDIT 10
10 FOR X = 20 to 24
RUN
 .7151002
 .683111
 .4821425
 .9992938
 .6465093
Ok
```

As long as a is positive, this sequence will be generated. The initial value of a has no effect.

The sequence of random numbers can be changed by specifying a value for a which is either negative or zero. If a is negative or zero, a new **seed** will be generated. The seed is the number which the computer uses to determine the initial value of the random number sequence. We can change the sequence of random numbers which will be generated by adding the following line to our program:

```
5 PRINT RND(-100)
```

Since RND's argument is negative, this statement will result in a new seed. A new series of random numbers will be generated by now executing our program:

```
Ok
LIST
 5 PRINT RND(-100)
10 FOR X = 1 TO 5
20 PRINT RND(X)
30 NEXT X
40 END
Ok
RUN
  .8188288
  .2677991
  8.733116E-02
  7.081251E-02
  .8175731
  .5208339
Ok
```

A new seed was generated by the execution of line 5. This caused new random numbers to be generated by lines 10 to 30.

The desired range for the random number is not always 0 to 1. The following program illustrates how to obtain a number in a different desired range which is, in this case, 1 to 6:

```
Ok
LIST
10 DI = INT(6 * RND + 1)
20 PRINT "You rolled a";DI
30 END
Ok
RUN
You rolled a 5
Ok
```

Since this program simulates the roll of a die, we needed a random integer value between 1 and 6, inclusive, not a random

number between 0 and 1. The randomly generated number which was returned in this case was .7151002. Because the maximum desired value was 6, the randomly generated number was multiplied by 6. The minimum desired value was 1, so 1 was added to the result of the last calculation. Finally, the number was truncated using INT.

RANDOMIZE

Each time the program in the last section is run, the same answer results. RANDOMIZE resets the seed each time the statement is executed. With each execution, the person running the program is asked to enter a seed. The number which is entered determines what value will be returned by RND. When RANDOMIZE is used in conjunction with RND, it is not necessary to give RND a seed. The following program illustrates the effect of RANDOMIZE:

```
Ok
LIST
10 RANDOMIZE
20 DI = INT(6 * RND + 1)
30 PRINT "You rolled a";DI
40 END
Ok
RUN
Random number seed (-32768 to 32767)? 927
You rolled a 4
Ok
```

We selected 927 as the seed. Any number between -32768 and 32676 will be accepted. The value for DI was then calculated, using 927 as the seed. Execution of line 40 causes that value to be output.

SCREEN

BASIC's SCREEN function returns the ASCII code for the character at a specified location on the screen. The format for the SCREEN function is:

SCREEN(row, column, true or false)

The row and column are required parameters. Rows are numbered from 0 to 24. 0 is the row at the top of the screen, and 24 is the number of the bottom row. The columns are numbered from 1 to 40, unless WIDTH is set to 80. If WIDTH is set to 80, the columns are numbered from 1 to 80. Column number 1 is the left edge of the screen, and column number 40 is the right edge.

The third parameter is optional. If the value of this parameter equals 0, or false, SCREEN returns the ASCII code for the character at the specified location. If the value does not equal 0, it is considered to be true, and the color is returned. The color range is from 0 to 255. The color MOD 16 is the foreground color, or the color of the character. The background color is the initial returned value MOD 128. If this value is tested to see if its value is greater than 127, a value of -1, or true, means that the character is flashing. A value of 0 indicates that it is not flashing.

The following example demonstrates the result of BASIC's SCREEN function:

```
Ok
PRINT SCREEN (2,7)
 83
Ok
PRINT CHR$(83)
S
Ok
```

In this example the character at (2,7) is "S". The ASCII code for "S" is 83.

User-Defined Functions

A user-defined function is a function which the programmer defines. Before a user-defined function can be called, it must be defined. The following command defines a function:

DEF FN rest of function name (dummy argument) = definition

An example of a function definition is:

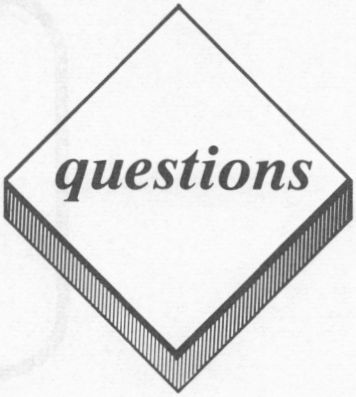
```
DEF FNTHREE(X) = (X ^ 3) - (3 * X)
```

DEF indicates that the function whose name follows is to be defined. FNTHREE is the function name. All user-defined function names must begin with FN. In this case, X is the dummy argument. When the function is called, the calling value is substituted for X. Wherever X appears within the definition, that value is substituted for X. Any valid variable name can be the dummy argument. If a data type has been specified for the dummy argument, the data will be converted to the dummy argument's type prior to substitution. If the data type is specified by the function name, the value returned by that function will be of the specified type.

In this example the user-defined function is a numeric function. User-defined functions can be either numeric or string. The definition, or set of operations which the function is to perform, in this example is (X ^ 3) - (3 * X). In other words, X is to be cubed. The value of 3 * X is then to be subtracted from the cube. The following program illustrates another user-defined function:

```
Ok
LIST
10 DEF FNRAISE(X) = X ^ X
20 FOR C = 1 TO 5
30 PRINT FNRAISE(C);
40 NEXT C
50 END
Ok
RUN
 1   4   27   256   3125
Ok
```

In this program the function name is FNRAISE. FNRAISE is a function name, not a variable. X is the dummy argument. Each time line 30 is executed, the function is called. Each number is then raised to itself. For example, 3 is cubed, and 5 is multiplied by itself 5 times. Note that reserved words, with the exception of FN, cannot be used as part of a valid function name.



True or False

1. When FRE is used with a string argument, housekeeping is performed during execution.
2. POS returns the row in which the cursor is located.
3. POKE returns the value in a specified memory location.
4. RANDOMIZE resets the seed.
5. If WIDTH is set to 40 when using SCREEN, the columns are numbered from 0 to 39.

Multiple Choice

1. Which of the following is a valid function name?

   A. FN
   B. FNNAME
   C. FNOKAY
   D. OKAY
   E. None of the above

2. RND returns a number within which of the following ranges?
   A. -1 to 1
   B. -1 to 0
   C. 0 to 1
   D. It depends upon the seed
   E. None of the above

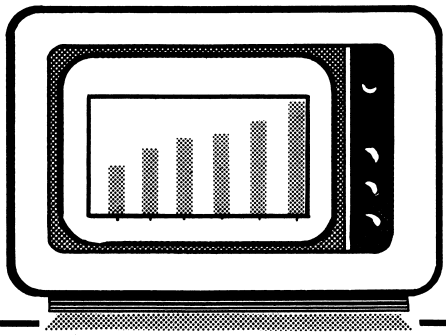
3. Which of the following commands specifically alters the contents of a specified memory location?
   A. PEEK
   B. POKE
   C. SCREEN
   D. FRE
   E. None of the above

4. When POS is executed and the WIDTH is set to 40, a number within which of the following ranges is returned?
   A. 0 to 39
   B. 1 to 40
   C. 0 to 24
   D. 1 to 25
   E. None of the above

5. What is the primary use of FRE?
   A. To define functions
   B. To look at a location in memory
   C. To alter a location in memory
   D. To determine how much memory is available
   E. None of the above

Computer Exercises

1. Write a program to randomly generate a number between 1 and 10. Use RND and RANDOMIZE.
2. After executing the program, write a program to do the following:
   a. Find how much space is available using FRE.
   b. Find the cursor's position.
   c. Determine the character in the tenth row, tenth column (9,10).



Introduction to Graphics

lesson 20

Lesson Goals

- *Define pixels and screen coordinates*
- *Define the differences between the various screen modes: text, low resolution graphics, medium resolution graphics, and high resolution graphics*
- *Learn the uses of BASIC's SCREEN, WINDOW, and VIEW statements*

Introduction

In this lesson we will introduce **graphics.** Graphics can be defined as the art of drawing with the computer. A number of programs, including many computer games, utilize graphics. In this lesson we will discuss the basics of graphics. In the next lesson we will discuss how to actually create pictures using the PC*jr*.

Pixels

The screen is divided into rows and columns. Rows are the horizontal dimension, and columns are vertical. The rows and columns are numbered. Every column intersects with every row, and each intersection is a specific and unique location on the screen. One such location would be the intersection of column 8 and row 11. The point where column 8 intersects row 11 is referred to as (8,11). Note that the column number precedes the row number and that the numbers are enclosed in parentheses. These numbers are known as **coordinates.**

Each specific coordinate pair references a **pixel.** Pixel is an abbreviation for picture element. A pixel is a small rectangular area located on the video screen. Each pixel can be referenced by its coordinates. A pixel is shown in figure 20.1.

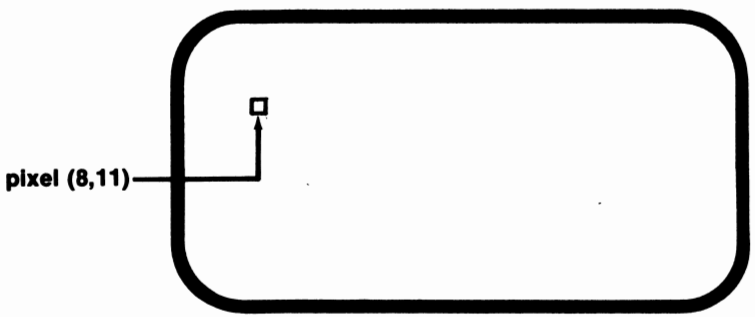


Figure 20.1. Pixel located at column 8, row 11

The numbering of the columns and rows begins with zero. The pixel which has the coordinates (0,0) is located in the top left-hand corner of the screen. Columns are numbered from left to right, and rows are numbered from top to bottom.

The specific number of rows and columns which are available is dependent upon the screen's **mode.** The mode refers to the form in which data is represented on the screen. The PC*jr* has one text mode and six graphics modes available. The PC*jr*'s predecessors, the PC and the PC XT, have only three modes available. These modes correspond to the first three PC*jr* modes. We will discuss the various modes later in this lesson.

The computer keeps track of which pixel was last referenced by means of an internal pointer. This pointer is called the LPR, or last point referenced.

Text Mode

The mode which we have been using throughout this book has been the text mode. Text mode is available only in the first mode, screen 0. Screen 0 divides the display into 25 rows and either 40 or 80 columns, depending upon the width.

Color can be displayed in this mode. Text mode has 8 options for the background color, 16 options for the border color, and 32 options for the color of the foreground, the characters. Table 20.1 lists the colors and their corresponding values.

Notice that the numbering begins with zero. Colors 0 through 7 are available for the background. Colors 0 through 15 can be used for the border area. Any color in the table can be used for the foreground. Colors 16 through 31 flash.

Table 20.1. Text mode color values

Text Mode Colors	
0 Black	16 Black, flashing
1 Blue	17 Blue, flashing
2 Green	18 Green, flashing
3 Cyan	19 Cyan, flashing
4 Red	20 Red, flashing
5 Magenta	21 Magenta, flashing
6 Brown	22 Brown, flashing
7 White	23 White, flashing
8 Gray	24 Gray, flashing
9 Lt. Blue	25 Lt. Blue, flashing
10 Lt. Green	26 Lt. Green, flashing
11 Lt. Cyan	27 Lt. Cyan, flashing
12 Lt. Red	28 Lt. Red, flashing
13 Lt. Magenta	29 Lt. Magenta, flashing
14 Yellow	30 Yellow, flashing
15 High Intensity White	31 High Intensity White, flashing

The COLOR statement is used to select colors. This command has three optional parameters:

COLOR foreground color, background color, border color

Any combination of parameters is permissible. If the new value for a parameter is omitted, the current color will be retained. Unless changed, both the background and the border will be black, color 0, and the characters will appear in white, color 7. Enter the following commands to demonstrate the use of BASIC's COLOR statement:

```
COLOR 5
COLOR ,5
COLOR ,,5
COLOR 7
COLOR ,0
COLOR ,,0
COLOR 21
COLOR 5
COLOR 7
```

The first command changes the foreground color to magenta. Entering the second command changes the background color to magenta, and the characters disappear. COLOR ,,5 creates a matching border. The command, COLOR 7, causes the foreground to reappear, since the characters are now displayed in white. The next two commands return the background and border to black. COLOR 21 changes the text so that it appears in magenta and flashes. Entering the command, COLOR 5, stops the flashing, and the final command returns the screen to its normal color configuration.

Low Resolution Graphics

Screen 3 is the only low resolution graphics mode. In low resolution graphics, the screen has few pixels compared to the other graphics modes. The screen's dimensions in low resolution graphics are 160 columns by 200 rows. The total number of pixels on the screen is equal to the number of columns times the number of rows, so 32000 pixels are available. Notice the illustration of the low resolution graphics screen in figure 20.2.

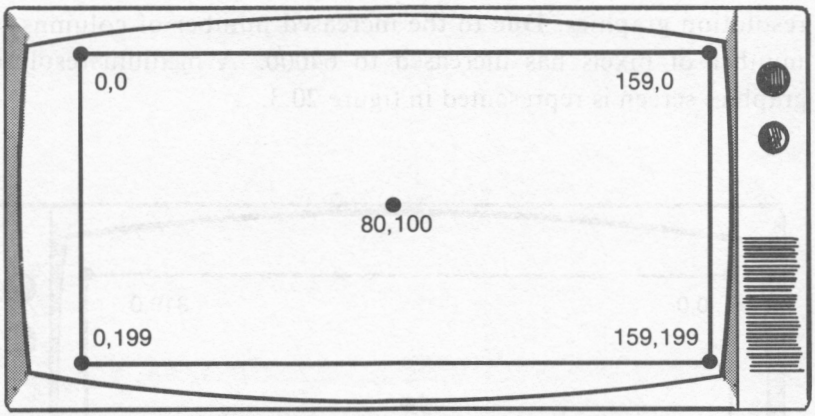


Figure 20.2. Low resolution graphics screen

The columns are numbered from 0 to 159, and the rows are numbered from 0 to 199. The center pixel of the screen has the coordinates (80,100). Whenever a graphics mode is first entered, the LPR is set to the screen's center pixel.

In low resolution graphics, sixteen colors are available, colors 0 through 15. In this mode, color 15 is the standard foreground color, and black is standard for the background. No border color can be set. Each low resolution graphics screen occupies 16K, or 16384 bytes of memory.

Text can be entered in any mode, although only one mode is specifically designed for text. In low resolution graphics, however, only 20 columns are available for text characters.

Medium Resolution Graphics

The PC*jr* has three medium resolution graphics modes: screen 1, screen 4, and screen 5. While each screen has its own features, they share some characteristics. A medium resolution graphics screen contains 320 columns, or twice that of the low resolution graphics

screen. The number of rows available, 200, is the same as in low resolution graphics. Due to the increased number of columns, the number of pixels has increased to 64000. A medium resolution graphics screen is represented in figure 20.3.

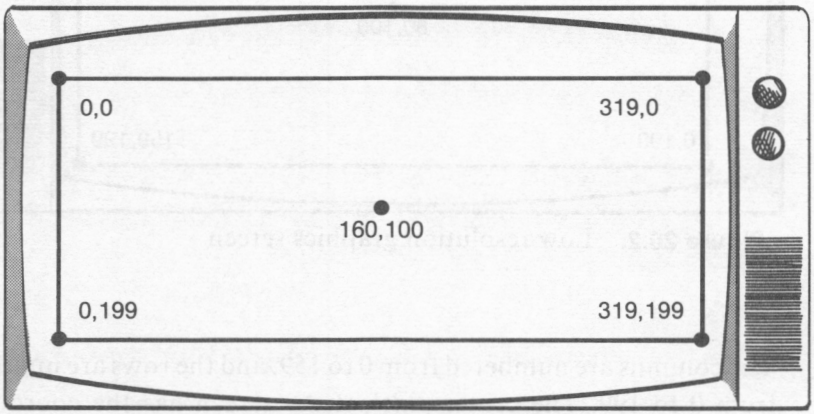


Figure 20.3. Medium resolution graphics screen

Notice that the center pixel in a medium resolution graphics screen is (160,100). In addition, all of the medium resolution graphics screens accommodate 40 characters of text per line.

Screen 1

Screen 1 is a medium resolution graphics screen. It requires 16K of memory and theoretically has 16 colors available. The colors, however, are available only in fixed palettes of four colors each. In practice then, only four colors are available at any time. The background color can be any of the 16 colors. Table 20.2 lists the palettes and their colors.

Table 20.2. Screen 1 palettes

Palette 0	Palette 1
0 background	0 background
1 green	1 cyan
2 red	2 magenta
3 brown	3 white

The COLOR command is used with a different configuration in screen 1 than in screen 0. By default, the color in the palette which is used for the foreground is assumed to be color 3. The COLOR statement's format is:

COLOR background color, palette

The parameters are optional. If no background color is specified, the color of the background will remain unchanged. The default value for the palette is the current palette. When screen 1 is first entered, the palette which is in use is palette 1.

Screen 4

Like screen 1, screen 4 requires 16K of memory and has 16 colors available. Screen 4's main advantage is that the palettes are flexible. Any four colors can be included in any palette. Screen 1 was primarily included in the PC*jr* so that much of the IBM PC software would be compatible.

In order to take advantage of the flexible palettes, a new command is needed. This command is PALETTE, which has the following configuration:

PALETTE location in palette, new color

This command is used to place colors in a palette. Red, for example, is usually located in position 2 of palette 0. PALETTE 2,1 will result in red being replaced by blue. Both parameters are required. Once the palette has been set, the format for the color command is:

COLOR foreground, background

Screen 5

Screens 5 and 6 require 32768 bytes of memory. The PC*jr* only allots 16384 bytes to the screen, however. An additional command is needed in order to allocate more memory to the screen. This command is CLEAR. CLEAR sets aside space in memory for the screen. CLEAR has the following configuration:

CLEAR,,,amount of space in memory to allocate to screen

The commas indicate optional parameters. These parameters are not relevant to our discussion at this point, so we will not discuss them here. In order to access screens 5 and 6, 32768 should be specified as the amount of memory to allocate.

In screen 5, all 16 colors are supported with no palettes. All 16 colors are accessible. The COLOR command has the same configuration in screen 5 as in screen 4.

High Resolution Graphics

Screens 2 and 6 are the high resolution graphics modes. In the high resolution graphics mode, the screen has 128000 pixels available. The rows are numbered from 0 to 199, and the columns are numbered

from 0 to 639. The center of the screen has the coordinates (320,100). In addition, screens 2 and 6 support 80 text columns. The following figure illustrates the high resolution graphics screen:

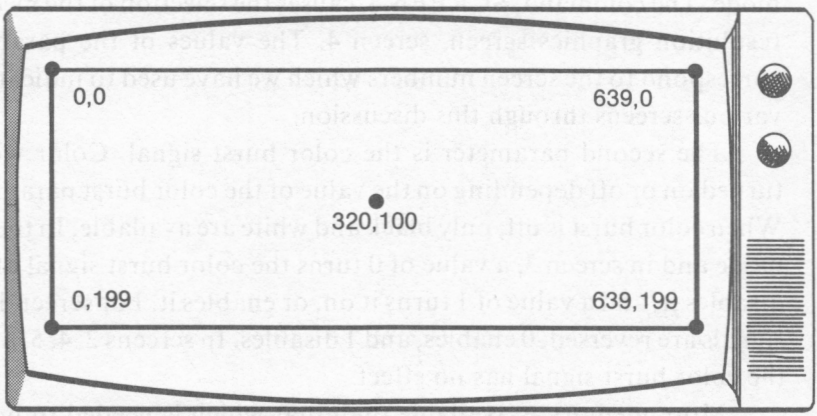


Figure 20.4. High resolution graphics screen

Screen 2 requires 16384 bytes of memory. Because of its high number of pixels, only two colors are supported. These colors are black and white. These colors cannot be changed. In contrast, screen 6 supports all 16 colors in a flexible palette. The COLOR command has the same configuration in screen 6 as in screens 5 and 4. Remember, however, that 32768 bytes of memory are required for this mode.

SCREEN

In the last lesson, we introduced BASIC's SCREEN function. In this section we will introduce BASIC's SCREEN command.

The SCREEN command is used to determine which of the 7 display modes is currently active. The command has the following configuration:

SCREEN screen number, color burst, active page, visual page, erase

The parameters are optional. The first parameter, the screen number, is used to select one of the six graphics modes or the text mode. The command, SCREEN 4, causes the selection of the medium resolution graphics screen, screen 4. The values of the parameter correspond to the screen numbers which we have used to indicate the various screens through this discussion.

The second parameter is the color burst signal. Color will be turned on or off depending on the value of the color burst parameter. When color burst is off, only black and white are available. In the text mode and in screen 3, a value of 0 turns the color burst signal off, or disables it, and a value of 1 turns it on, or enables it. For screen 1, the signals are reversed; 0 enables, and 1 disables. In screens 2, 4, 5, and 6, the color burst signal has no effect.

More memory is available than that which is needed to hold a single screen, or **page.** In other words, the contents of more than one screen can be held in memory at the same time. In the text mode, for example, up to 8 pages, numbered 0 through 7, can be retained in memory, assuming width has been set to 40. The key factors are the amount of memory alloted to the screen and the amount of memory needed to store one page. Since each page in text mode, which has a width of 40, requires 2K of memory, 8 pages can be held.

The next two parameters relate to the PC*jr*'s ability to store several pages in memory. The first of these parameters, the **active page,** is used to indicate to which page output is to be sent. The second of these parameters is used to specify which page is to be displayed upon the screen. This page is the **visual page.**

The final parameter is erase. Its value should be an integer between 0 and 2, inclusive. A value of 0 indicates that the video memory should not be erased. A value of 1 is the default value. It indicates that the video memory should be erased if the mode or burst signal is changed. Finally, a value of two indicates that all video memory should be erased if the mode or burst changes.

Entering the following commands will illustrate the uses of the SCREEN command:

```
SCREEN,,,1
SCREEN,,1
SCREEN,1
SCREEN 1
COLOR 9,1
SCREEN 2
SCREEN 3
COLOR 8,4
SCREEN 4
PALETTE 2,1
COLOR 2,1
CLEAR,,,32768
SCREEN 5
SCREEN 6
```

By default the screen is set to 0,0,0,0,1. In other words, after booting the PC*jr*, the screen is in the text mode, color is disabled, both the active and visual pages are page 0, and video memory will be erased if the mode or burst changes. The first command changes the visual page to page 1. The second command transfers the output to page 1. SCREEN, 1 disables the color. The command , SCREEN 1, changes the mode to medium resolution graphics. The next command, COLOR 9,1 selects 9, light blue, as the background color and also selects palette 1 for the foreground. After that command, the next two commands change the mode to screens 2 and 3. Once in screen 3, the command, COLOR 8, 4 selects gray, 8, as the foreground color and red, 4, as the background color. SCREEN 4 selects the medium resolution graphics mode of screen 4. The command, PALETTE 2,1, replaces the red with blue in palette 0. The next command causes blue to be the foreground color and green to be the background color. The remaining commands change the mode to screens 5 and 6.

WINDOW

This command only operates in cartridge BASIC. WINDOW is used to reset the coordinates of the screen. The command has the following configuration:

WINDOW(first column number, first row number)-(last column, last row)

The first pair of numbers is the coordinates to be assigned to the pixel which is located in the top left-hand corner of the screen. The second pair is used to define the coordinates to be assigned to the pixel in the bottom right-hand corner. Any numbers can be used as coordinates.

VIEW

VIEW is also available in cartridge BASIC. This command defines an area of the screen to which the graphics will be limited. This area is known as a **viewport.** Once a viewport has been defined, points can only be plotted within its boundaries. The format for the command is:

VIEW(first column,first row)-(last column,last row)

The first coordinate pair is the location of the top left-hand pixel in the viewport. The second pair identifies the pixel in the bottom right-hand corner of the viewport.

A drawing will automatically be scaled to fit into the viewport. If you don't want the drawing to be scaled, the following command will define a viewport and inhibit the scaling:

VIEW SCREEN(first column,first row)-(last column,last row)

Ppaint color, boundary color

The paint parameter is the number of the color which is to be used to fill in the figure. If the active mode uses palettes, the range is limited to 0 to 3. In those modes which support 16 colors, the range is 1 to 15. The default value for this parameter is the background color. The boundary color parameter specifies the color of the figure's border. When that color is encountered, the painting will halt. While both parameters are necessary for the process to be completed correctly, only the boundary color is required. The following program illustrates the use of this command:

```
Ok
LIST
10 SCREEN 4:CLS
20 DRAW "C3 U50 R20 D50 L20"
30 DRAW "C3 BU50 L30 U20 R80 D20 L50"
40 DRAW "BF10"
50 DRAW "P1,3"
60 DRAW "BU15"
70 DRAW "P2,3"
80 END
Ok
RUN
```

The commands contained in lines 20 and 30 draw two closed figures. Execution of line 40 moves the LPR to a pixel in the interior of the bottom figure. In order for P to execute properly, the LPR must be pointing to a pixel within the interior of the figure. Line 50 is then executed, causing the figure to be filled with color 1. The LPR is relocated to the interior of the second figure in line 60, and that figure is filled with color 2 when line 70 executes.

It is a good idea to use C to specify the color in which to plot the figure. When that same color is specified as the boundary color, the

TAangle measurement in degrees

A positive value for the angle measurement rotates the figure counterclockwise. A negative value rotates the figure clockwise. Possible angle measurements include -123, 96, and 11. The value must be an integer. The following program illustrates the use of TA:

```
Ok
LIST
10 SCREEN 2:CLS
20 DRAW "BU80"
30 DRAW "TA-69 R305"
40 DRAW "TA150 R300"
50 DRAW "TA0 R300"
60 DRAW "TA-150 R300"
70 DRAW "TA69 R305"
80 END
Ok
RUN
```

When line 20 is executed, the LPR is relocated. During the execution of line 30, the angle rotates clockwise 69 degrees and performs the specified movement. The next line, line 40, rotates the angle 150 degrees and draws the specified movement. Lines 50 through 70 continue the process.

Unless a new value for TA is indicated, all subsequent movements will be drawn based on TA's value. If, for example, TA150 had not been designated in line 40, the movement R300 would have extended the line drawn in line 30 an additional 300 pixels.

P

P is used to fill a closed figure with color. This GDL command has the following format:

inclusive. That value is multiplied by 90 in order to obtain the number of degrees to rotate. A value of 0 indicates that the movements should be executed normally. A value of 1, however, indicates that the movements should be rotated 90 degrees. If the rotational angle is 1, the movement R20 will resemble the movement U20 when the rotational angle for that movement is set to 0. If the value specified is 2, the movements will be rotated 180 degrees. A value of 3 causes a 270 degree rotation. The effect of A can be seen more clearly by running the following program:

```
Ok
LIST
10 CLEAR,,,32768
20 SCREEN 5:CLS
30 DRAW "A1 R60 U10 M+15,+10 M-15,+10 U10 BL60"
40 DRAW "A2 R60 U10 M+15,+10 M-15,+10 U10 BL60"
50 DRAW "A3 R60 U10 M+15,+10 M-15,+10 U10 BL60"
60 DRAW "A0 R60 U10 M+15,+10 M-15,+10 U10 BL60"
70 END
Ok
RUN
```

Notice that the commands in lines 30 through 60 are virtually identical. The only difference is the value associated with A. In line 30 that value is 1, so the movements are rotated 90 degrees and plotted. In line 60 the value is 0. This specification was required in order to cancel the rotation of 270 degrees which was created in line 50. Once a rotational angle is specified, all subsequent movements will be plotted according to the identified rotation.

TA

TA rotates the figure. Unlike GDL's A command, TA can rotate the figure any number of degrees within the range of +360 and –360. The command has the following configuration:

allotted range for the foreground of the selected graphics mode. For example, since screen 1 uses fixed palettes, the color number should be 1, 2, or 3. Three, in this case, is the default value. The C command only sets the foreground color. It does not modify either the background color or the border color. This point is illustrated in the following modification of the previous program:

```
Ok
LIST
10 SCREEN 3:CLS
20 DRAW "C6 NU60 ND60 NL60 NR60"
30 DRAW "C5 NE40 NF40 NG40 NH40"
40 FOR J = 1 TO 1000:NEXT J
50 END
Ok
RUN
```

When line 20 was executed, the foreground color was changed to color 6, brown, and the movements were plotted in that color. Movements in subsequent statements will also be plotted in that color, unless a new color is specified. In line 30 a new color, magenta, is specified, and those movements are plotted in color 5.

A

The GDL command used to select a rotational angle is A. This command has the following format:

Arotational angle

The rotational angle is specified by an integer between 0 and 3,

N

Like B, N always precedes a GDL command. N plots the specified movement but leaves the LPR unchanged. An example of a command which utilizes N is DRAW "NR30". Again, no spaces or punctuation marks should separate N from the GDL movement command. The program which follows illustrates the use of N:

```
Ok
LIST
10 SCREEN 3:CLS
20 DRAW "NU60 ND60 NL60 NR60"
30 DRAW "NE40 NF40 NG40 NH40"
40 FOR J = 1 TO 1000:NEXT J
50 END
Ok
RUN
```

Notice that each movement is plotted. Since N precedes each movement command, however, the LPR is left at the screen's midpoint.

C

C sets the color in which the points are to be plotted. This GDL command has the following format:

Ccolor number

C's parameter, the color number, can be any color number within the

B

B is always specified with a GDL movement command. B indicates that a movement should be made, but the pixels involved should not be plotted. In other words, after B is executed, the LPR will point to the specified pixel, but the movement to that pixel will not have been plotted. Since B is a prefix, it must precede a GDL movement command. An example is DRAW "BM35,70". No spaces or punctuation marks should separate B and the movement command. The following program illustrates the use of B:

```
Ok
LIST
 10 SCREEN 4:CLS
 20 DRAW "BM60,85"
 30 DRAW "F40 BM100,85"
 40 DRAW "G40 BM110,85"
 50 DRAW "F12 D28 BM122,97"
 60 DRAW "E12 BM147,85"
 70 DRAW "R30 M147,125 R30 BM157,105"
 80 DRAW "R10"
 90 FOR J = 1 TO 1000:NEXT J
100 END
Ok
RUN
```

When line 20 is executed, the LPR is moved to (60,85) but that movement is not plotted. When line 30 is executed, a diagonal line is drawn and the LPR is subsequently relocated to (100,85). Line 40 draws an intersecting diagonal line and moves the LPR to (110,85). This process creates a gap between this figure and the figure which is drawn by lines 50 and 60. When line 60 is executed, the LPR is moved to (147,85). This movement creates a gap between the second and third figures.

M

M is another movement option. Movement occurs from the LPR to any specific screen location. The screen location may be indicated in either absolute or relative form. The command has one of the following formats depending upon the chosen coordinate form:

M column, row
or
M ±column, ±row

The sign, which indicates the change in direction, must precede the column number and row number if relative form is selected. A STEP statement is not needed. The following program demonstrates some uses of M:

```
Ok
LIST
10 CLEAR,,,32768
20 SCREEN 6:CLS
30 DRAW "M290,80"
40 DRAW "M+30,-50"
50 DRAW "M350,80"
60 DRAW "M-30,+20"
70 DRAW "M320,30"
80 LINE(290,80)-(350,80),,,15422
90 END
Ok
RUN
```

Lines 30, 50, and 70 illustrate the use of the absolute form. In contrast, lines 40 and 60 use the relative form. A + before the column number indicates movement to the right, and a – directs movement to the left. When working with rows, a – indicates upward movement, and a + indicates movement downward.

Scaling Factor

The scaling factor can be set by using the GDL's S command. S has the following format:

Sscale

The scale can be any integer within the range of 1 to 255. The scale divided by 4 is the actual scaling factor. The standard scaling factor is 1, so the normal value for scale is 4. Notice what happens when line 20 is edited to include the characters S8:

```
Ok
LIST
 10 CLEAR,,,32768:SCREEN 6:CLS
 20 DRAW "S8 E50 H50 G50 F50"
 30 FOR K = 1 TO 2
 40 DRAW "H10 E50"
 50 DRAW "H10 G50"
 60 NEXT K
 70 DRAW "H10"
 80 FOR L = 1 TO 2
 90 DRAW "E10 F50"
100 DRAW "E10 H50"
110 NEXT L
120 END
Ok
RUN
```

Remember that the value associated with S previously was 4. When the scale was changed to 8 in line 20, the scale was changed not only for line 20, but also for all subsequent lines.

Diagonal Movements

GDL includes four commands for diagonal movement. These commands are E, F, G, and H. The following formats apply to the diagonal commands:

Edistance
Fdistance
Gdistance
Hdistance

The distance is the number of pixels to be plotted in the specified direction. E indicates a move up and to the right. F moves down and to the right. G directs movement down and to the left, and H indicates a move up and to the left. The following program demonstrates the use of these commands:

```
Ok
LIST
 10 CLEAR,,,32768:SCREEN 6:CLS
 20 DRAW "E50 H50 G50 F50"
 30 FOR K = 1 TO 2
 40 DRAW "H10 E50"
 50 DRAW "H10 G50"
 60 NEXT K
 70 DRAW "H10"
 80 FOR L = 1 TO 2
 90 DRAW "E10 F50"
100 DRAW "E10 H50"
110 NEXT L
120 END
Ok
RUN
```

When line 20 executes, a diamond is created. Lines 30 to 110 draw a series of intersecting diagonal lines. Note that the aspect ratio does not affect diagonal movement.

Vertical and Horizontal Movements

GDL has four commands which belong in this category: U, D, L, and R. These commands have the following formats:

Udistance
Ddistance
Ldistance
Rdistance

No space or punctuation mark separates the distance from the command. The distance is a whole number which indicates how many pixels are to be plotted in the specified direction.

The commands move vertically and horizontally. U moves upward, D moves down, L moves to the left, and R moves to the right. The following program illustrates the use of these commands:

```
Ok
LIST
10 CLEAR,,,32768
20 SCREEN 6:CLS
30 DRAW "L160 D90 R160 U90"
40 DRAW "L140 D90"
50 FOR B = 115 TO 185 STEP 5
60 LINE(160,B)-(320,B)
70 NEXT B
80 END
Ok
RUN
```

When line 30 executes, a rectangle is drawn. Line 40 plots a vertical line. Execution of lines 50 to 70 causes a series of horizontal lines to be drawn. Note that four horizontal movements occupy the same space as three vertical movements.

```
Ok
LIST
 10 CLEAR,,,32768
 20 SCREEN 6:CLS
 30 CIRCLE(320,100),100,,,,.55
 40 CIRCLE(320,130),35,2,-3.14,-6.28,.25
 50 CIRCLE(320,90),25,,-4.44,-5.14,1
 60 CIRCLE(285,83),15,,,,.3
 70 CIRCLE(355,83),15,,,,.3
 80 CIRCLE(285,79),20,,.78,2.34
 90 CIRCLE(355,79),20,,.78,2.34
100 CIRCLE(355,83),5,,,,1
110 CIRCLE(285,83),5,,,,1
120 FOR J = 1 TO 1000:NEXT J
130 END
Ok
RUN
```

When line 30 is executed, a large oval is drawn. During the execution of line 40, a long semicircle is drawn in color 2. Note that the end points are connected to the center point by lines. Line 50 draws an arc whose end points are connected by a line to its center point. When lines 60 and 70 are executed, smaller ellipses are plotted. Execution of lines 80 and 90 causes two arcs to be drawn. Lines 100 and 110 add two small ellipses to the picture.

DRAW

DRAW creates an object whose dimensions are defined by a string of graphics commands. The commands are part of the **Graphics Definition Language,** or GDL™. When a DRAW command is executed, each GDL command in the string is executed separately. These commands draw lines, set angles, set colors, set scales, and execute substrings. The DRAW command has the following format:

DRAW "string of drawing commands"

CIRCLE(center point coordinates),radius,color,starting angle,
ending angle,aspect

CIRCLE STEP(center point coordinates)radius,color,starting angle,
ending angle,aspect

The coordinates of the center point are required. Any pixel on the screen can serve as the center point for the circle. The coordinates again may be in either absolute or relative form.

The length of the radius is also required. The radius is a straight line which extends from the center point of a circle to a point on the circle's edge. This value must be positive, but it does not have to be an integer.

Color is the first optional parameter. If the color is left unspecified, the foreground color will be used.

The next set of optional parameters are the starting and ending angles. The allowed range for these angles is 2 * PI to -2 * PI. These parameters are used to define the beginning and end of semicircles and arcs. If the angles are negative, the end points of the arc will be connected to the center point with a line. The angles themselves are always considered to be positive.

The final optional parameter is the **aspect.** The aspect is the ratio of height to width. Unless specified, the ratio is 9:5 or 1.8 in a low resolution mode, 5:6 or .833333 in the medium resolution modes, and 5:12 or .4166667 in the high resolution modes. The shape of the circle is defined by the aspect ratio. If the given aspect ratio is larger than the default value, the ellipse is stretched vertically. If the specified aspect value is less than the default value, the ellipse is stretched horizontally. The following program illustrates some of the possibilities of the CIRCLE statement:

```
Ok
LIST
 10 SCREEN 4:CLS
 20 LINE(140,80)-(180,120),,B
 30 LINE(140,80)-(155,65)
 40 LINE-STEP(40,0)
 50 LINE-(180,80)
 60 LINE(195,65)-(195,105)
 70 LINE-(180,120)
 80 LINE(155,65)-(155,80)
 90 LINE(140,120)-(155,105),1,,15422
100 LINE-(155,81),1,,15422
110 LINE(155,105)-(195,105),1,,15422
120 FOR J = 1 TO 1000:NEXT J
130 END
Ok
RUN
```

Line 10 clears screen 4. Execution of line 20 draws a box. When line 30 is executed, a line is drawn from one specified point to another specified point. Lines 40 and 50 also draw lines, but the starting points for the lines are the LPR. During the execution of lines 60, 70, and 80, three more lines are drawn. When line 90 is executed, another line is drawn, but this line is in a different color and a different style. Lines 100 and 110, when executed, draw lines similar to those drawn by line 90.

CIRCLE

BASIC's CIRCLE statement can be used to draw circles, semi-circles, ellipses, and arcs. The command may have either of the following formats:

LINE(starting point coordinates)-(ending point coordinates),color,
B or BF, style

or

LINE STEP(starting point coordinates)-STEP(ending point coordinates),
color, B or BF, style

The coordinates of the starting point are optional. If the starting coordinates are not specified, the initial point is assumed to be the LPR. If LINE is being used to draw a box rather than a line, these coordinates should identify a corner of the box. Note that the coordinates may be in absolute or relative form.

The coordinates of the end point are required. This pair of coordinates is preceded by a mandatory -. Again, the coordinates may be in either relative or absolute form. If the figure being drawn is a box, this set of coordinates should indicate the opposite corner of the box.

The next parameter is the color. As with PSET, its default value is the foreground color. This parameter is optional.

If a line is being drawn, the next parameter, B or BF, is omitted. If a box is being drawn, however, this parameter is required. The parameter has two options: B and BF. The B indicates that a box is to be drawn. BF indicates that the box is to be drawn and filled in with the color which is in use. If F is used, it must be preceded by B with no comma or space separating the two letters.

The final option is style. The style is the pattern in which the line is to be drawn. If style 0 is selected, no points are plotted. When any other style is chosen, only certain points will be plotted. Which points will be plotted is dependent upon the pattern specified by the style. Each style has a different pattern. The style can be any integer between -32768 and 32767. The following program demonstrates the use of LINE and its parameters:

The previous program can be modified to include PRESET. By adding line 155, the effect of PRESET can be illustrated. Notice that in line 155, relative form coordinates are used:

```
Ok
LIST
 10 SCREEN 4:CLS
 20 FOR B = 80 TO 140
 30 PSET(100,B)
 40 NEXT B
 50 FOR A = 101 TO 116
 60 PSET(A,113)
 70 NEXT A
 80 FOR C = 113 TO 140
 90 PSET(117,C)
100 NEXT C
110 PSET(130,96),1
120 FOR D = 113 TO 140
130 PSET(130,D),1
140 NEXT D
150 FOR J = 1 TO 1000:NEXT J
155 PRESET STEP(0,-44)
160 END
Ok
RUN
```

When line 155 is executed, the point appears to be erased. If a color other than the background color is specified for PRESET, its function is similar to PSET. The point will be plotted in the specified color.

LINE

BASIC's LINE statement can be used to draw lines or rectangular shapes, known as boxes. LINE has either of the following configurations:

```
Ok
LIST
 10 SCREEN 4:CLS
 20 FOR B = 80 TO 140
 30 PSET(100,B)
 40 NEXT B
 50 FOR A = 101 TO 116
 60 PSET(A,113)
 70 NEXT A
 80 FOR C = 113 TO 140
 90 PSET(117,C)
100 NEXT C
110 PSET(130,96),1
120 FOR D = 113 TO 140
130 PSET(130,D),1
140 NEXT D
150 FOR J = 1 TO 1000:NEXT J
160 END
Ok
RUN
```

During execution of this program each pixel is individually plotted. The pixel at (100,80) is plotted first. Then the pixel at (100,81) is plotted. The points plotted by the execution of lines 30, 60, and 90 are in color 3. The points plotted by lines 110 and 130 are in color 1. If the color had not been specified in line 130, those points would have been plotted in color 3.

In contrast to PSET, the default for PRESET is color 0, the background color. When the background color is used to plot the pixel, the point appears to be erased. The PRESET command can have either of the following formats:

PRESET(absolute form coordinates), color

or

PRESET STEP(relative form coordinates), color

This phrase, however, is not used alone. It is executed jointly with other graphics statements. In many instances where absolute form coordinates are used, relative form coordinates may also be used. This point will be illustrated more concretely in the following sections.

PSET and PRESET

PSET and PRESET plot individual pixels. PSET draws a point at a specified screen location. This command may have either of the following formats:

PSET(column, row), color
or
PSET STEP(column change, row change), color

The coordinates are required, but the color is an optional parameter. If the color is not specified, the foreground color is used to plot the pixel. Remember that the standard foreground colors are: 15 in screens 3 and 5; 3 in screens 1, 4, and 6; and 1 in screen 2. The following program illustrates the use of PSET:

Introduction

In lesson 20 we explored the underlying concepts of graphics. In this lesson we will use those concepts in conjunction with several of Microsoft BASIC's graphics statements to create pictures. BASIC has several graphics statements available.

It is a particularly good idea when reading this lesson to enter and execute the sample programs. Since a black and white book cannot reproduce the PC*jr*'s graphics, executing the programs will enable you to actually see the effects of the various statements.

Absolute and Relative Form

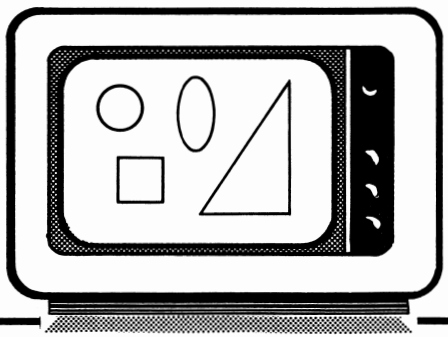
Throughout our discussion, each pixel has been referenced by its actual screen coordinates. This type of referencing is known as **absolute form.** Another method for referencing pixels is to specify a location based on the LPR's location. Pixels identified in this manner are considered to be in **relative form,** because exactly which pixel is being referenced is dependent upon the LPR. The format for relative form coordinates is:

(change in column, change in row)

Suppose the LPR is pointing to the pixel which has the coordinates (80,100). The pixel at (70,100) is located ten columns to the left and ten rows below the LPR. This pixel can be referenced by the absolute form coordinates (70,110) or by the relative form coordinates of (-10,10).

Relative form coordinates are used in conjunction with STEP. The following phrase is used to indicate relative form:

STEP(change in column, change in row)



Graphics Statements

lesson 21

Lesson Goals

- *Learn how to use BASIC's PSET and PRESET statements to plot individual pixels*
- *Learn how to use BASIC's LINE statement to draw lines and boxes*
- *Learn how to use BASIC's CIRCLE statement to draw circles, ellipses, semicircles, and arcs*
- *Learn how to use BASIC's DRAW statement to create shapes*
- *Learn how to use PAINT to add colors and patterns to figures*

2. Which of the following types of graphics has 128,000 pixels?
   A. High resolution graphics
   B. Low resolution graphics
   C. Medium resolution graphics
   D. All of the above
   E. None of the above

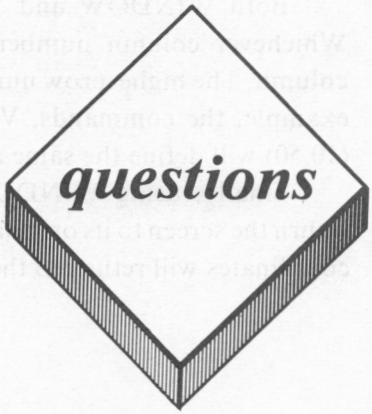
3. Which of the following screen uses low resolution graphics?
   A. Screen 0
   B. Screen 1
   C. Screen 3
   D. Screen 6
   E. Screen 7

4. Assuming the amount of memory allocated to the screen is its default value, how many pages of 40 column text can be held in text mode?
   A. 1
   B. 4
   C. 8
   D. 16
   E. None of the above

5. How many colors are available for the background in text mode?
   A. 8
   B. 16
   C. 2
   D. 4
   E. None of the above

Essays

1. Discuss the relative advantages of the different screens and graphics modes.



True or False

1. The PC*jr* has 6 graphics modes.
2. SCREEN can only be used to select the display mode.
3. When working with the color burst signal, 1 always turns the signal off.
4. In the text mode, 16 options are available for the background color.
5. WINDOW and VIEW will order coordinate pairs.

Multiple Choice

1. Which of the following coordinate pairs is located in the top left-hand corner of the screen in medium resolution graphics?

   A. (0,0)
   B. (0,199)
   C. (319,0)
   D. (319,199)
   E. None of the above

Both WINDOW and VIEW will order the coordinate pairs. Whichever column number is lower will be considered the first column. The highest row number will be considered the last row. For example, the commands, VIEW(10,30)–(20,50) and VIEW(20,30)-(10,50) will define the same area.

Finally, using WINDOW or VIEW without parameters will return the screen to its original state. Viewports will be erased, and the coordinates will return to their initial values.

probability that the command will have the desired effect increases. If this command does not seem to be executing properly, you may want to check whether the LPR points to a pixel inside the closed figure and whether the boundary color is encountered.

X

GDL's X command allows a substring to be executed from within another string. The command has the following configuration:

Xvariable name;

Notice that this command is followed by a semicolon. The variable name identifies the substring. The variable name does not include the initial X.

This command allows a portion of a figure to be defined separately from the entire figure. If drawing a house, for example, the roof can be defined separately from the house by using X. X can also be useful when you are drawing a complex figure which requires more than 255 characters to complete its definition. Remember that the maximum number of characters per line is 255. The following program demonstrates the use of X:

```
Ok
LIST
 10 CLEAR,,,32768
 20 SCREEN 6:CLS:KEY OFF
 30 WALL$ = "BM10,198 R40 BR150 R429 U196 L619 D96"
 40 DOOR$ = "BR40 TA35 R150"
 50 SOFA$ = "TA0 BM520,148 NR109 U106 L229 U40"
 60 COFTAB$ = "BM350,60 R100 D75 L100 U75"
 70 CHAIR$ = "BM100,2 M10,50 M110,90 M200,40 M100,2"
 80 TV$ = "BM10,100 R60 D50 L60"
 90 DRAW "XWALL$;XDOOR$;XSOFA$;XCOFTAB$;XCHAIR$;XTV$;"
100 END
Ok
RUN
```

Each substring draws a separate part of the floor plan. In lines 30 through 80, different GDL commands are assigned to variables. The graphics commands are not executed until line 90.

In line 20 we introduced a new statement, KEY OFF. KEY OFF simply removes the messages which appear at the bottom of the screen.

GDL Commands and Variables

The numeric values associated with the GDL commands may be variables or constants. Throughout this discussion, we have used constants, but variables are also a valid option. The format for using a GDL command with a variable is as follows:

GDLcommand = variable name;

The graphics command is followed by = and the variable name. A semicolon must be placed after the variable name. The following program illustrates the use of variables with GDL commands:

```
Ok
LIST
10 SCREEN 1:CLS
20 FOR SCALE = 1 TO 30
30 COL = SCALE MOD 3 + 1
40 DRAW "C=COL;S=SCALE;U10;R10;D10;L10;"
50 NEXT SCALE
60 END
Ok
RUN
```

Each time line 40 is executed, the scale increases and the color changes. Note that the commands in line 40 are separated by semi-

colons. After a variable or an X command, a semicolon is required. Under other conditions the semicolon is optional.

PAINT

The PAINT statement is not available in cassette BASIC. PAINT fills a selected area of the screen with color. The statement has the following format:

PAINT(coordinates of an interior pixel),color,boundary,background

The coordinates may be in either absolute or relative form. If the indicated pixel is within a closed figure, the painting process will halt when the border is encountered. If the pixel is located on the border of the figure, no painting will occur. Finally, if the pixel is located outside of a closed figure, the unenclosed areas of the screen will be filled with the specified color.

Color is the color to be used to fill in the figure. Color is an optional parameter. If it is omitted, the color selected is the color with the highest value supported by the active mode: 1, 3, or 15. The value for color may also be a string expression. If the color value is a string expression, tiling will be performed. We will discuss tiling later in this section.

Boundary specifies the color of the edges of the figure which is to be painted. This parameter is optional. If the boundary color is not specified, the paint color is assumed to be the boundary color. The painting process will continue until the boundary color is encountered. If the boundary color is not encountered, the entire screen will be painted.

The background is a string expression used in tiling to invalidate a termination condition. We will explore this optional parameter in more depth when we discuss tiling.

The following program illustrates the use of PAINT without tiling:

```
Ok
LIST
 10 CLEAR,,,32768
 20 SCREEN 6
 30 CLS
 40 CIRCLE(320,120),130,,-3.14,-6.28,.18
 50 LINE(370,120)-(374,40),,BF
 60 DRAW "M250,100 R120"
 70 PAINT(300,90),3,3
 80 PAINT(320,130),2,3
 90 PAINT(300,110),1,3
100 END
Ok
RUN
```

When line 40 is executed, a semicircle is drawn. Line 50 draws a box, and execution of line 60 completes a triangle. When line 70 is executed, the triangle is filled with color 3. Execution of line 80 results in the semicircle being painted in color 2. Finally, since the pixel specified by line 90 is outside of a closed area, the remaining areas of the screen are painted in color 1.

If a complex shape is to be painted, a large amount of space in memory may be required by the PAINT statement. It is a good idea to include a CLEAR statement at the beginning of any program in which complicated figures are to be painted.

Solid, one color patterns are not the only options available with PAINT. **Tiling** can be used to create patterns of varying design and color.

To use tiling, the color must be specified as a string of values rather than as a numeric value. This string is obtained by using hexadecimal numbers to designate a binary pattern. The form of this parameter is:

CHR$(&Hhexadecimal)+CHR$(&Hhexadecimal)+CHR$(&Hhexadecimal)

Each hexadecimal number is preceded by &H and has two places. The value contained in these two places requires up to 8 places in order to be represented in binary, or base two. The following table illustrates hexadecimal and binary conversion of the decimal value, 255:

Table 21.1. Hexadecimal and binary conversion

System	Base	Representation	Base 10 equivalent
Hexadecimal	16	FF	15 * 16 + 15 * 1 = 255
Decimal	10	255	2 * 100 + 5 * 10 + 5 * 1 = 255
Binary	2	11111111	1 * 128 + 1 * 64 + 1 * 32 + 1 * 16 + 1 * 8 + 1 * 4 + 1 * 2 + 1 * 1 = 255

The decimal number 255 is FF in base 16. This value is the equivalent of the binary number 11111111. The decimal value 255 is equal to (2 * 100) + (5 * 10) + (5 * 1).

The same principle applies to the process of converting a hexadecimal number to its decimal equivalent. The rightmost position is still the one's place, but the second position is the sixteen's place. The next position is the 256's place. In order to derive the decimal equivalent for a number in base 16, the value in each position is multiplied by the value of the place. Fifteen, the value of F, multiplied by one, the rightmost position's value, is equal to 15, and 15 multiplied by 16, the next position's value, yields 240. The sum of 15 and 240 is 255, the decimal equivalent of FF.

Binary-decimal conversion follows the same principle. The positions, moving from right to left, increase in value by powers of two. In order to obtain the corresponding decimal value, the value of that place is multiplied by the value in that position, and the values are summed.

Each hexadecimal number in the CHR$ expressions is converted to its binary equivalent by the BASIC interpreter. Eight places are required in binary to represent a value which requires two hexadecimal places. Values larger than the decimal value 255 cannot be used for tiling, since only eight binary places are allotted per CHR$ expression. When working with tiling, each of these eight-place binary numbers is called a **tile mask.** Each binary place represents the contents of one bit of memory, so eight bits of memory are represented by each tile mask. As we discussed in lesson 2, eight bits of memory are known as one byte. Each CHR$ expression therefore represents the contents of one byte of memory. In the high resolution graphics mode of screen 2, one bit of memory is required to plot one pixel. Consequently, each tile mask represents the contents of eight pixels. If a 1 is located in the 2^2 or fours place, the pixel which is indicated by that position will be plotted. If a 0 occupies that position, the pixel will not be plotted.

As many as 64 CHR$ expressions can be listed as the color in a single PAINT statement. These expressions will be recycled until the area is completely tiled. If, for example, ten bytes of data are needed to paint the area, but only 4 CHR$ expressions are supplied, the first, fifth, and ninth bytes will be indicated by the first CHR$ expression, and the second, sixth, and tenth bytes will be plotted according to the values contained in the second CHR$ expression. This pattern will continue until the process is complete.

The following program uses PAINT's tiling feature to fill a circle with a series of Z's:

```
Ok
LIST
10 SCREEN 2:CLS
20 CIRCLE(320,100),100
30 PAINT(320,100),CHR$(&HFF)+CHR$(&H2)+
CHR$(&H4)+CHR$(&H8)+CHR$(&H10)+CHR$(&H20)
+CHR$(&H40)+CHR$(&HFF)
40 END
Ok
RUN
```

When line 30 executes, the CHR$ expressions are converted to their binary equivalents. The following table lists the conversion equivalents of the tile masks:

Table 21.2. Conversion equivalents

Tile byte	Hexadecimal	Decimal	Binary
1	FF	255	11111111
2	2	2	00000010
3	4	4	00000100
4	8	8	00001000
5	10	16	00010000
6	20	32	00100000
7	40	64	01000000
8	FF	255	11111111

Notice that the pattern of ones in the binary equivalents is identical to the pattern of pixels which were plotted.

In screen 2 only one bit of data is required to plot one pixel. The other screens require more information before a pixel can be plotted. When either screen 1, 4 or 6 is active, for instance, 2 bits of data are needed for each pixel. Hence, 4 pixels are plotted by each CHR$ expression. The extra data identifies the color, 1 to 3, to be used. Table 21.3 shows the binary and hexadecimal values associated with colors 1 through 3 in any palette.

Table 21.3. Palette patterns

Color No.	Number in binary	Binary pattern to draw solid line	Hexadecimal
1	01	01010101	&H55
2	10	10101010	&HAA
3	11	11111111	&HFF

Screens 3 and 5 have 16 colors available. Since the number of options for the color has increased, the number of bits of information needed in order to plot one pixel also increases. Only two pixels are plotted by each CHR$ expression. Table 21.4 lists the patterns required to draw a solid line in screens 3 and 5.

Table 21.4. Line patterns

Color No.	Number in binary	Binary pattern to draw solid line	Hexadecimal
1	0001	00010001	&H11
2	0010	00100010	&H22
3	0011	00110011	&H33
4	0100	01000100	&H44
5	0101	01010101	&H55
6	0110	01100110	&H66
7	0111	01110111	&H77
8	1000	10001000	&H88
9	1001	10011001	&H99
10	1010	10101010	&HAA
11	1011	10111011	&HBB
12	1100	11001100	&HCC
13	1101	11011101	&HDD
14	1110	11101110	&HEE
15	1111	11111111	&HFF

The following program paints a circle in screen 5 with solid lines of all 15 colors:

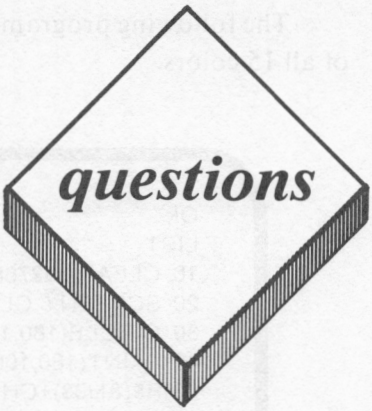
```
Ok
LIST
10 CLEAR,,,32768
20 SCREEN 5:CLS
30 CIRCLE(180,100),100
40 PAINT(180,100),CHR$(&H11)+CHR$(&H22)+
CHR$(&H33)+CHR$(&H44)+CHR$(&H55)+CHR$(&H66)
+CHR$(&H77)+CHR$(&H88)+CHR$(&H99)+CHR$(&HAA)
+CHR$(&HBB)+CHR$(&HCC)+CHR$(&HDD)+CHR$
(&HEE)+CHR$(&HFF)
50 END
Ok
RUN
```

When line 30 is executed, a circle is drawn. As line 40 executes, the circle is painted with lines of color.

The background tile is used to specify a line pattern which you do not want the computer to consider as a termination condition. This optional parameter has the following format:

CHR$(&Hhexadecimal)

This pattern will not be considered as a possible termination pattern. Up to two line patterns may be specified. The standard termination condition occurs when two identical lines are encountered. When this condition is met, tiling is halted.



True or False

1. PSET and PRESET have the same default color values.
2. BASIC's CIRCLE statement cannot be used with relative form coordinates.
3. DRAW uses GDL commands.
4. GDL commands and statements such as LINE may not appear in the same program.
5. N plots movement but does not change the LPR.

Multiple Choice

1. Which of the following commands is not a GDL command?
   A. N
   B. P
   C. LINE
   D. M
   E. None of the above

2. Which of the following commands is not valid?
   A. DRAW "BL30"
   B. DRAW "E20"
   C. DRAW "P1"
   D. DRAW "R= VARDISTANCE;"
   E. None of the above

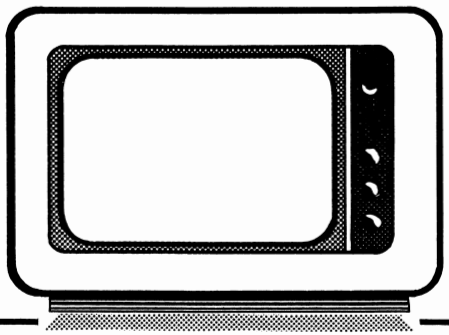
3. Which of the following values is equal to the decimal value, 134?
   A. 86 base 16
   B. 10000110 base 2
   C. 134 base 16
   D. A and B
   E. None of the above

4. What is the maximum number of CHR$ expressions which can be included in a PAINT statement as part of the color parameter?
   A. 1
   B. 8
   C. 63
   D. 64
   E. None of the above

5. When working with LINE, what is the default value for the color?
   A. The foreground color
   B. The background color
   C. The border color
   D. The style
   E. None of the above

Computer Exercises

1. Write a program to draw a house. Use as many of the various graphics commands as practical.



Introduction to Sound

lesson 22

Lesson Goals

- *Gain an understanding of the PCjr's mechanisms for sound generation*
- *Learn how to create sound using BASIC's SOUND command*
- *Learn how to use BASIC's PLAY command to create sound*
- *Learn how to use BASIC's NOISE command to produce sound effects*

Introduction

In this lesson we will learn how to create sounds and musical tones with the PC*jr*. We will begin our exploration with a discussion of the PC*jr*'s capacities and the mechanisms which are responsible for sound generation. After this discussion we will create sounds using the PC*jr*.

Sound Generators

The PC*jr* has two separate mechanisms for generating sound. One, the 8253 timer, is common to both the PC*jr* and the PC. The other mechanism is unique to the PC*jr*. This internal mechanism is Texas Instrument's programmable tone generator, the SN 76489A. With the SN 76489A, the number of options for the creation of sound is greatly increased.

The 8253 timer is capable of producing one note at a time. The duration, volume, and pitch of that note can be programmed. The timer can use any of three mechanisms in order to output the sound: the PC*jr*'s internal speaker, an audio jack of a stereo, or a television set's speaker. The audio jack and the television set are considered to be external outputs. The inclusion of the timer in the PC*jr* enables it to emulate the PC.

The programmable tone generator, however, can accomplish the entire range of timer capabilities in addition to its own capacities. The SN 76489A can generate three notes of differing volumes and durations at the same time. It is also capable of producing sound effects. Since the tone generator is more flexible than the timer, our discussion will focus on the former. The tone generator, however, only uses external outputs, since the PC*jr*'s internal speaker is not capable of handling that many tones at the same time.

SOUND

BASIC's SOUND statement is used to activate either the timer or the programmable tone generator, depending upon the statement's configuration. The command may have either of the following formats:

SOUND ON
or
SOUND OFF

If SOUND is set to on, the programmable tone generator is activated, and the internal speaker is switched off. When SOUND is off, both the timer and the internal speaker are activated. OFF is the default value for SOUND.

BEEP

BEEP is used to activate the external speakers. BEEP may have either of the following configurations:

BEEP ON
or
BEEP OFF

BEEP ON is the default situation. When BEEP is on, the external speakers are activated. If BEEP is set to off, the external speakers are switched off.

It seems that when SOUND is on and BEEP is off, no sound is possible. This is due to the fact that the internal speaker is disabled by the status of the SOUND command, and the external speakers are deactivated by the BEEP OFF command. If this combination is in operation, however, the external speakers are activated despite the command to the contrary.

SOUND

The SOUND statement has an alternate configuration which is used to actually create sounds. This configuration is as follows:

SOUND pitch, duration, volume, voice

The pitch is the frequency in Hertz. The pitch can be any number within the range of 37 to 32767. This number does not have to be an integer. If the value specified is less than 110, the frequency 110 Hz will be generated, since 110 Hz is the lowest frequency which the PC*jr* can support. A value of 32767 will create silence. If the pitch is 14000 or higher, we won't be able to hear it.

Each musical note has a corresponding frequency. When working with SOUND, that frequency must be specified. Table 22.1 lists some common notes and their frequencies.

Table 22.1. Note-frequency list

Note	Frequency	Note	Frequency
C	261.63	middle C	523.25
C#	277.18	C#	554.37
D	293.66	D	587.33
D#	311.13	D#	622.25
E	329.63	E	659.26
F	349.23	F	698.46
F#	369.99	F#	739.99
G	392.00	G	783.99
G#	415.30	G#	830.61
A	440.00	A	880.00
A#	466.16	A#	932.33
B	493.88	B	987.77

In order to raise a note one octave, double that note's frequency. To lower the note one octave, halve the frequency. Pitch is a required parameter.

The duration is also required. The duration is the length of time a note or a rest is to be sustained. One second is equivalent to 18.2 duration counts. If you want a note to be sustained for three seconds, the value of the duration parameter should be 54.6. If the value specified is zero, the note will not be played.

Volume is an optional parameter. The value can be any number within the range of 0 to 15, inclusive. Numbers with decimal portions will be rounded to integers. If the specified value is 0, no sound will be generated. Fifteen is both the loudest option and the default value.

The final optional parameter is voice. A voice is a mechanism which can produce any one note. The PC*jr* has three tonal voices and one noise voice. With SOUND only the three tonal voices can be accessed. This parameter can have an integer value of 0, 1, or 2. Zero is the default value. The following program uses SOUND to play a scale:

```
Ok
LIST
 10 SOUND ON
 20 BEEP ON
 30 SOUND 523.25,2
 40 SOUND 587.33,2,13
 50 SOUND 659.26,2,11
 60 SOUND 698.46,2,9
 70 SOUND 783.99,2,11
 80 SOUND 880.00,2,13
 90 SOUND 987.77,2,15
100 END
Ok
RUN
```

This program produces the tones of the octave which begins at middle C. Line 10 activates the tone generator. Line 20 is not required

since, by default, the external speakers are activated. Lines 30 through 90 create the tones. Duration is constant, but the volume varies. Only one voice, voice 0, was used in this program. If the voices had been mixed, chords would have been generated instead of single notes.

When Cassette BASIC is active, only one sound at a time is produced. After each SOUND statement, execution continues normally until another SOUND statement is encountered. Execution pauses until the first sound is completed. Then the second sound is executed. If this new sound has a duration of zero, the voice is quieted immediately.

Cartridge BASIC, on the other had, has the capacity to buffer, or store in memory, up to 32 unplayed notes. Execution is continued even when a new SOUND is encountered. If a tone with a duration of zero is buffered, an error will result.

PLAY

In order to activate all three voices at once, three SOUND statements would be required. While this method is effective, it is not efficient. BASIC's PLAY statement is more efficient. The statement's configuration is as follows:

```
PLAY music0$,music1$,music2$
```

Music0$, music1$, and music2$ are string constants or expressions of music commands. These music commands are similar to the GDL commands used with DRAW, so they have been referred to as the "tune definition language." The first string controls voice 0, the second string controls voice 1, and the third string controls voice 2. If any string is omitted, that voice is not used.

Notes

Unlike SOUND which relies on frequencies, PLAY uses the actual notes. This command has the following format:

Note sharp or flat

The note is a letter between A and G, inclusive. The note may be followed by a symbol indicating sharp or flat. Either # or + may be used to indicate a sharp note, and a - may be used to denote a flat note. The following commands illustrate the use of PLAY using one, two, and three voices:

```
Ok
SOUND ON
Ok
PLAY "D F# DC+ F# C+"
Ok
PLAY "CDE","EFG"
Ok
PLAY "CDE","EFG","GAB"
Ok
```

SOUND ON activated the tone generator. The second command used both sharp symbols in order to play the first two measures of "The Band Played On." In the third command, two voices were used, and in the final command all three voices were used to create very simple chords. The voices may be used in any combination.

L

L is used to set the length of the notes. If L is unspecified, as in our previous example, quarter notes will be generated. This com-

mand has the following format:

L number

The number is required and may be any integer within the range of 1 to 64. A value of 1 indicates that the note is to be a whole note, a value 2 indicates a half note, and a value of 4 denotes a quarter note. If L is specified, all subsequent notes will have that value until L is changed. If you want to change the length of only one note, follow that note by the value. An example is A8. In this case, A will be played as an eighth note. This change in A's duration will not affect the duration of other notes. The following program demonstrates the use of L:

```
Ok
LIST
10 SOUND ON
20 PLAY "L4 F8 F8 FC"
30 PLAY "L8 AAA4 F4 FA"
40 END
Ok
RUN
```

This program plays the opening notes of "Clementine." In line 20, length is set to quarter notes, but the first two notes were played as eighth notes. In line 30, length is set to 8, so eighth notes are the default value.

MB and MF

MB plays music in the background. In other words, MB activates the tone generator's buffering capability. MB is the standard value. In contrast, MF deactivates the buffer. Running the following program will illustrate the effects of MB and MF:

```
Ok
LIST
10 SOUND ON
20 PLAY "MF CD E2 E2"
30 PLAY "C2 DEF","E2 FGA"
40 FOR K = 1 TO 2000:NEXT K
50 PLAY "MB L4 C D E2 E2"
60 PLAY "C2 DEF","E2 FGA"
70 END
Ok
RUN
```

This program plays the first notes of "My Old Kentucky Home." Lines 20 and 30 are executed with music in the foreground. In other words, line 20 must be completed prior to line 30 being played. In lines 50 and 60, however, the buffer has been activated. Line 60's voice 1 begins playing before it should, creating some strange sound combinations.

Articulation

Articulation is the proportion of a note's length which is actually played. With Microsoft BASIC, three articulation options exist: ML, MN, and MS. ML stands for music legato. This command indicates that each note is to be played its entire length. No break between notes occurs.

MN is the default value. MN stands for music normal. Each note is played for .875 times its length. A very short break separates the notes.

Finally, MS is short for music staccato. With music staccato, each note is played for .75 * its length. The notes are clearly separated. The following program demonstrates the distinctions between the three commands:

```
Ok
LIST
10 SOUND ON
20 PLAY "ML D8 D D8 E G B2B2"
30 PLAY "MN D8 D D8 E G A1"
40 PLAY "MS D8 D D8 E B A2A2"
50 END
Ok
RUN
```

This program plays part of "Tom Dooley." Line 20 has music legato. Each note blends in with the next note. MN is specified in line 30, causing the notes to sound similar to their previous status. Finally, in line 40, music staccato is specified, and each note is played very distinctly.

N

The notes do not have to be identified by letters. They may also be identified by numbers which correspond to the keys of a piano. N has the following format:

N note number

The note number falls within the range of 0 to 84. A value of zero denotes a rest. The note with a value of 1 corresponds to the lowest key on the piano, and note 84 corresponds to the highest key. Middle

C has a value of 37. The following program replaces the notes specified with letters in the last example with note numbers:

```
Ok
LIST
10 SOUND ON
20 PLAY "ML L8 N51 L4N51 L8 N51 L4N53N56 L2N60N60"
30 PLAY "MN L8 N51 L4N51 L8N51 L4N53N56 L1N58"
40 PLAY "MS L8 N51 L4N51 L8N51 L4N53N60 L2N58N58"
50 END
Ok
RUN
```

In this program, nothing changed except for the system of identifying notes. Because of this change, however, the length had to be reset using L each time the note length changed. The numbers that specify length can only be used after letter notes.

O

O sets the octave. This command has the following structure:

O octave number

The octave number is a number between 0 and 7 which represents the octave in which the notes are to be played. Octaves range from C to B, and octave three has middle C as its initial note. The following program demonstrates the effects of O:

```
Ok
LIST
10 SOUND ON
20 PLAY "O3 EEAAA2B O4C2"
30 PLAY "O3 BO4C2O3AG1EG1"
40 END
Ok
RUN
```

This program plays the beginning part of "When Johnny Comes Marching Home." Since not all the notes are in the same octave, whenever the octave changes, that change must be specified.

P

P denotes a pause. This command has the following format:

P length

The length can range from 1 to 64 and operates the same as it does with notes. The following program demonstrates the use of P:

```
Ok
LIST
10 SOUND ON
20 PLAY "O2 B O3CDE1DEABEDC O2 A2 P4"
30 PLAY "CEFG1 A GE CED1 P4"
40 PLAY "O2 B CDE1 DEAGEDC O2A2P4"
50 END
Ok
RUN
```

The tune played by this program is part of "Londonderry Air." Each line ends with a pause. In this case, the pauses are all quarter rests. Notice that the length must be specified.

T

T sets the tempo. The tempo is the number of quarter notes per minute. The default value for T is 120, but T's value may be any integer between the range of 32 and 255. T has the following format:

T number of quarter notes per minute

The following program illustrates the effect of varying the value specified with T:

```
Ok
LIST
10 SOUND ON
20 FOR TEMPO = 50 TO 250 STEP 100
30 PLAY "T = TEMPO;"
40 PLAY "F8 F8FCA8A8A FF8A8 O4CC8 O3 B-AG2"
50 FOR K = 1 TO 2000:NEXT K
60 NEXT TEMPO
70 END
Ok
RUN
```

This program assigns to T the values 50, 150, and 250. The value 250 is the fastest. Notice that the value assigned to commands with PLAY does not have to be a constant. Variables are valid, as long as they are preceded by an equal sign and followed by a semicolon, as in line 30.

V

V is used to adjust the volume. This command is only valid if SOUND is on. V has the following format:

V volume level

The volume level can be any integer value which is included in the range of 0 to 15. The default value for V is 15. The next program demonstrates the effect of V:

```
Ok
LIST
10 SOUND ON
20 FOR VOL = 5 TO 15 STEP 5
30 PLAY "V = VOL;"
40 PLAY "O3 G2 GAB2 B O4 CD2 E D O3 B1"
50 PLAY "O4D2C O3 B A1 O4C2 O3 B A G1"
60 NEXT VOL
70 END
Ok
RUN
```

With each cycle of the loop, the volume increases. This program plays part of "Long Long Ago." Notice that VOL is a variable, not a constant.

Dotted Notes

A period following a note indicates that the note is to be played as a dotted note. In other words, its length should be multiplied by 1.5. C. is an example of a dotted note. A note may be dotted several times; C.... is an example of this possibility. In this case, C's length is multiplied 5.065 times. 5.065 is equal to 1.5 raised to the fourth power. The following program demonstrates the use of dotted notes:

```
Ok
LIST
10 SOUND ON
20 PLAY "O3 GGB O4DG2. E2. CDED1."
30 PLAY "O3GGB O4DD2. O3A2BO4CO3BAG1"
40 END
Ok
RUN
```

This program plays part of the familiar tune, "On Top of Old Smokey." In line 20, G and E are dotted. In line 30, D is the only dotted note. Pauses may also be dotted.

Changing Octaves

Earlier in this lesson we discussed the use of O to change octaves. The > and < signs may also be used to change octaves. The > is used to raise a note one octave. The < is used to lower a note one octave. These commands have the following formats:

<note
>note

The following program demonstrates the use of these symbols to change octaves:

```
Ok
LIST
10 SOUND ON
20 PLAY "DC8<B-8>D8C8<B->B-G8B-."
30 PLAY "F2D<B->C1D2L8C<B->DC"
40 PLAY "L4<B->B-G8B-FD8<B-8>CC8C8<B-1"
50 PLAY ">A.B-8>C<FF.G8FB-B-GE-GF1"
60 PLAY "D2L8C<B->DCL4<B->BG8B-."
70 PLAY "FG8B-8>CC8C8<B-1"
80 END
Ok
RUN
```

This program plays the melody line of "Swanee." The third note of the song is the B flat below middle C. The < sign is used to indicate that the B flat belongs to the lower octave. The D, however, is back in the original octave; the > changes the octave back. Each time the < sign appears, the octave is lowered, and with each appearance of >, the octave is raised. Notice that the octave is not changed until the next > or < appears.

X

X executes a substring, as it does in graphics. Its function and format are the same with both DRAW and PLAY. The following program demonstrates the use of X to create sound:

```
Ok
LIST
10 SOUND ON
20 ROW1$ = "L4 CCC8D8E"
30 ROW2$ = "L8 EDEFG2"
40 ROW3$ = "L4 >C<GEC"
50 ROW4$ = "L8 GFEDC2"
60 SONG$ = ROW1$ + ROW2$ + ROW3$ + ROW4$
70 SONG$ = SONG$ + SONG$
80 PLAY "O3 XSONG$;","O4P1 XSONG$;","O5 P1
    P1 XSONG$;"
90 END
Ok
RUN
```

This program plays "Row, Row, Row Your Boat" in a round. ROW1$, ROW2$, ROW3$, and ROW4$ store the song's melody. In line 60, the different parts of the melody are concatenated to form SONG$. In line 70 SONG$ is changed so that it will play twice. When line 80 is executed, the first voice begins the song in octave 3. After a whole rest, voice 1 begins the song in octave 4. One measure later, voice 2 begins the song in octave 5.

Noise

BASIC's NOISE command uses the noise voice to create sound effects. The noise voice has 8 types of sound, divided into two categories, periodic and white. Sounds 0-3 are periodic sounds. Periodic sounds are ragged, like the sound of a chain saw. Sounds 4 through 7 are the white sounds. White sounds hiss.

The NOISE command has the following format:

NOISEsource, volume, duration

The source is an integer in the range from 0 to 7. The source corresponds to the sound numbers. Table 22.2 lists the sources.

Table 22.2. Sound table

Periodic	White	Source	Sound
0	4	2330 Hz	high, less coarse
1	5	1165 Hz	medium
2	6	582 Hz	low, more coarse
3	7	voice 2 freq.	dependent upon voice 2's frequency

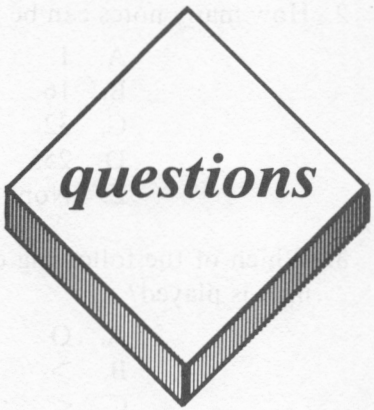
Sources 3 and 7 will vary in sound depending upon the frequency of the tonal voice, voice 2. The other sources use fixed frequencies and thus have fixed effects.

Volume and duration operate similarly in the NOISE and SOUND commands. Volume ranges from 0 to 15, and duration can vary. All three parameters are required.

The following program demonstrates the possibilities of the NOISE statement:

```
Ok
LIST
10 SOUND ON
20 FOR TYPE = 0 TO 7
30 CLS
40 PRINT "SOURCE #";TYPE
50 NOISE TYPE,15,25
60 FOR K = 1 TO 2000:NEXT K
70 NEXT TYPE
80 END
Ok
RUN
```

This program generates a sample of the sound produced by each of the sources. Notice the variation in pitch and type of sound. Volume and duration are constant.



True or False

1. The programmable tone generator has four voices.
2. L4 indicates that subsequent notes should be held for 4 counts.
3. X executes a substring.
4. The octave which contains middle C is octave two.
5. MF activates the tone generator's buffering capability.

Multiple Choice

1. Which of the following symbols may be used to sharp a note?
   A. #
   B. –
   C. +
   D. Both A and C
   E. None of the above

2. How many notes can be held in the buffer?

   A. 1
   B. 16
   C. 32
   D. 255
   E. None of the above

3. Which of the following commands affects the octave in which a note is played?

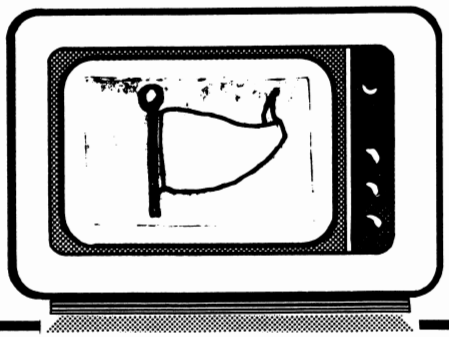
   A. O
   B. >
   C. <
   D. All of the above
   E. None of the above

4. Which of the following commands denotes a rest?

   A. R
   B. S
   C. T
   D. B
   E. None of the above

Computer Exercises

1. Write a program which generates a song. Use as many of the sound commands as practical.



Programming Techniques

lesson 23

Lesson Goals

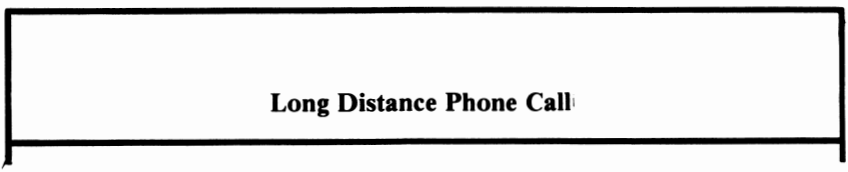
- *Learn the top-down design approach to programming*
- *Explore the use of menu-driven programs and delay routines*
- *Gain an understanding of programming techniques which utilize variables, including initializing variables, using variables as flags, and using significant variable names*

Introduction

In this lesson no new Microsoft BASIC commands will be introduced. Instead, we will explore a number of programming techniques which will enable you to make better use of those commands which you already know.

Top-Down Design

Top-down design is a process by which a programming problem is analyzed and then divided into smaller, more manageable pieces. Top-down design is merely a concrete elaboration of the thought processes which everyone uses, in one form or another, to solve problems. As an example, suppose that we want to describe how to place a long distance phone call to a friend. We decide to make a list of the steps which are involved. We can begin by writing the list's title at the top of a piece of paper:



The next step in the formation of the list is to decide upon categories into which the more detailed processes can be grouped. The following three categories provide the desired structure:

1. Find the telephone number
2. Dial the telephone
3. Use the telephone

We can then add these categories to the sheet of paper:

Long Distance Phone Call		
Find the telephone number	Dial the telephone	Use the telephone

Now that we have decided upon the general categories, we can begin to add details to the list. First we look at the left-hand category. Find the telephone number is relatively clear. More details, however, can be specified under Dial the telephone, the second category:

1. Dial 1
2. If necessary, dial the area code
3. Dial the telephone number

After adding these steps, the list has the following appearance:

Long Distance Phone Call		
Find the telephone number	Dial the telephone —Dial 1 —If necessary, dial the area code —Dial the telephone number	Use the telephone

Now we can turn our attention to the last category. Three subdivisions of this area also seem possible:

1. Wait for an answer
2. If answered, ask for friend
3. If friend is present, speak to friend

Once these steps are added, the list is complete:

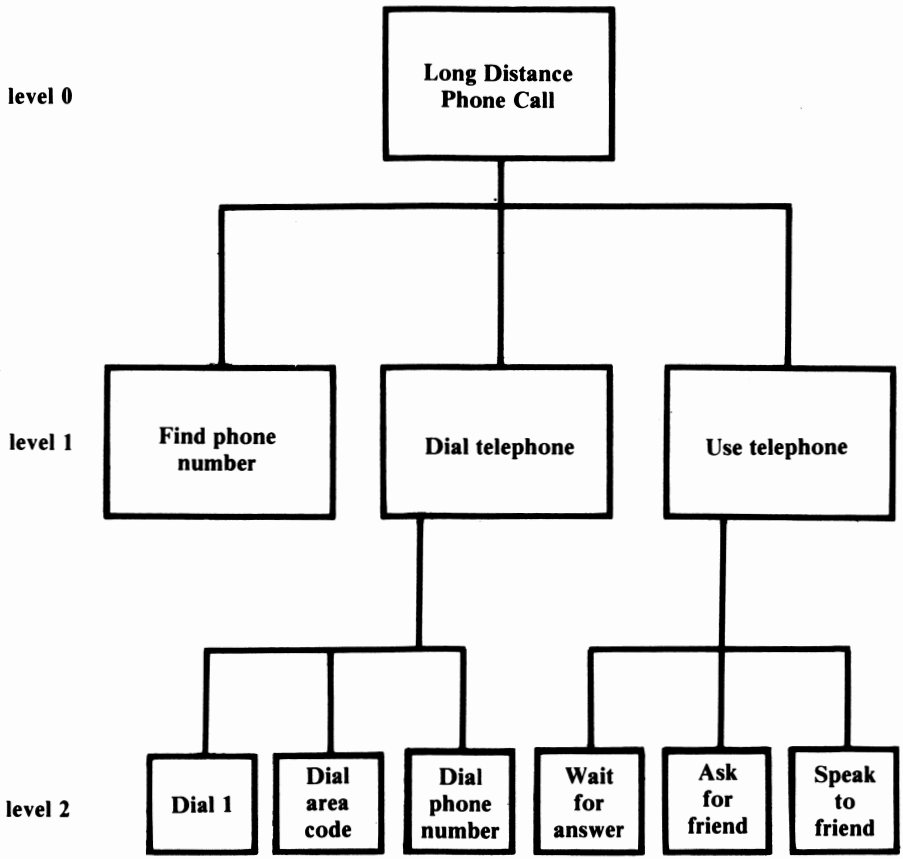
Long Distance Phone Call		
Find the telephone number	Dial the telephone —Dial 1 —If necessary, dial the area code —Dial the telephone number	Use the telephone —Wait for an answer —If answered, ask for friend —If friend is present, speak to friend

The same type of reasoning is used to create a top-down design. The title is the level 0 module. A level 0 module is a broad specification of the general intent. In this case, Long Distance Phone Call is comparable to a level 0 module.

The three categories, Find the telephone number, Dial the telephone, and Use the telephone, are the equivalents of level 1 modules. A level 1 module is a more detailed division of a level 0 module. The category Find the telephone number is complete at this level.

The other two categories, however, are further specified. Each division is the equivalent of a level 2 top-down design module. A level 2 module is called only by the level 1 module of which it is a subdivision. In other words, Dial the telephone number can only be accessed by Dial the telephone.

Let's assume the list was created from a top-down design. The list can then be depicted visually as a top-down design chart, such as the following:



Lines connect the various steps involved in making the call. One module can only invoke another module if the second module resides at a lower level and is connected to it by a line. These lower level modules are detailed divisions of the higher level modules.

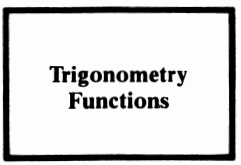
The actual structure of a top-down design is largely a matter of style. As a result, two people will usually create different top-down designs. This is especially true with complex problems.

Programmers generally find top-down design to be a very effective technique. By dividing a larger problem into successively smaller components, the problem can be more easily handled. Details are less overwhelming since attention is focused on a segment of the problem instead of on the entire problem. It is also easier to derive structure and order for a program when a top-down design is created first.

A top-down design should be completed prior to writing the program. When a programming problem seems simple, it is tempting to skip this preliminary step. It is not, however, a good idea. Any time which is saved by eliminating the top-down design step will most likely be required to eliminate the errors in the program. Creating a top-down design usually minimizes errors.

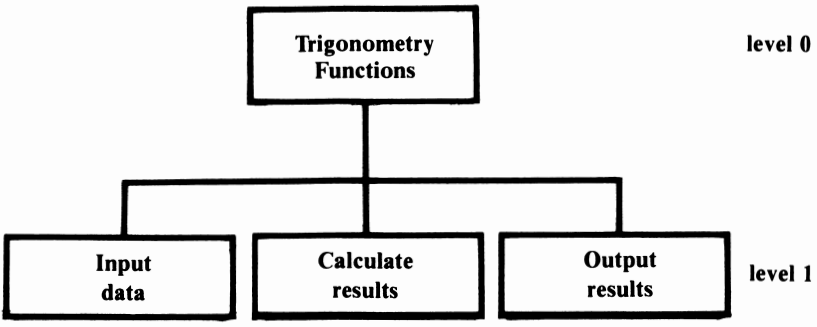
To illustrate the use of a top-down design approach, let's examine an actual programming example. Suppose we need to write a program which will accept either the degree or radian measure of an angle. The program should then output the sine, cosine, and tangent of that angle.

The first step is to create a top-down design. We begin by writing a name for the program. One possible name is Trigonometry Functions. We then diagram this step:

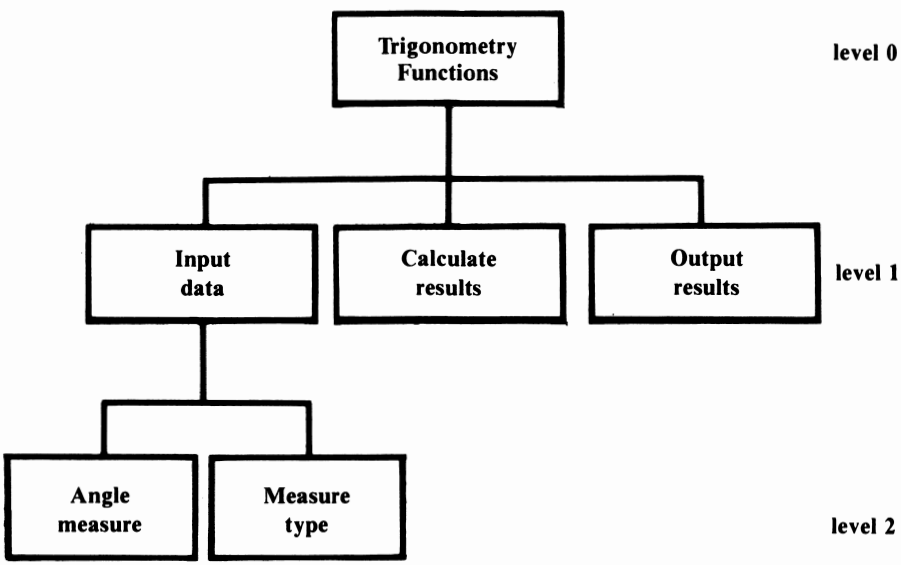


level 0

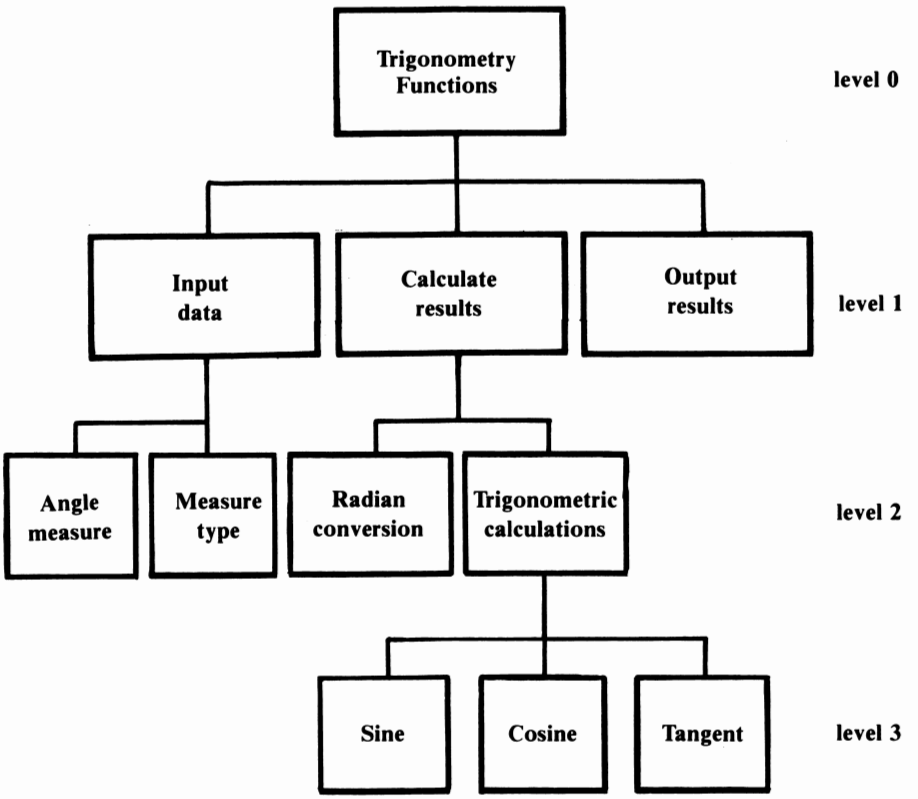
The problem is not yet clear. More details are required, so we look at the problem definition. This program is to accept the measure of an angle. This implies that the data needs to be input. The program is also supposed to output the results. Before the results can be output, they will need to be calculated. These three categories are the level 1 modules:



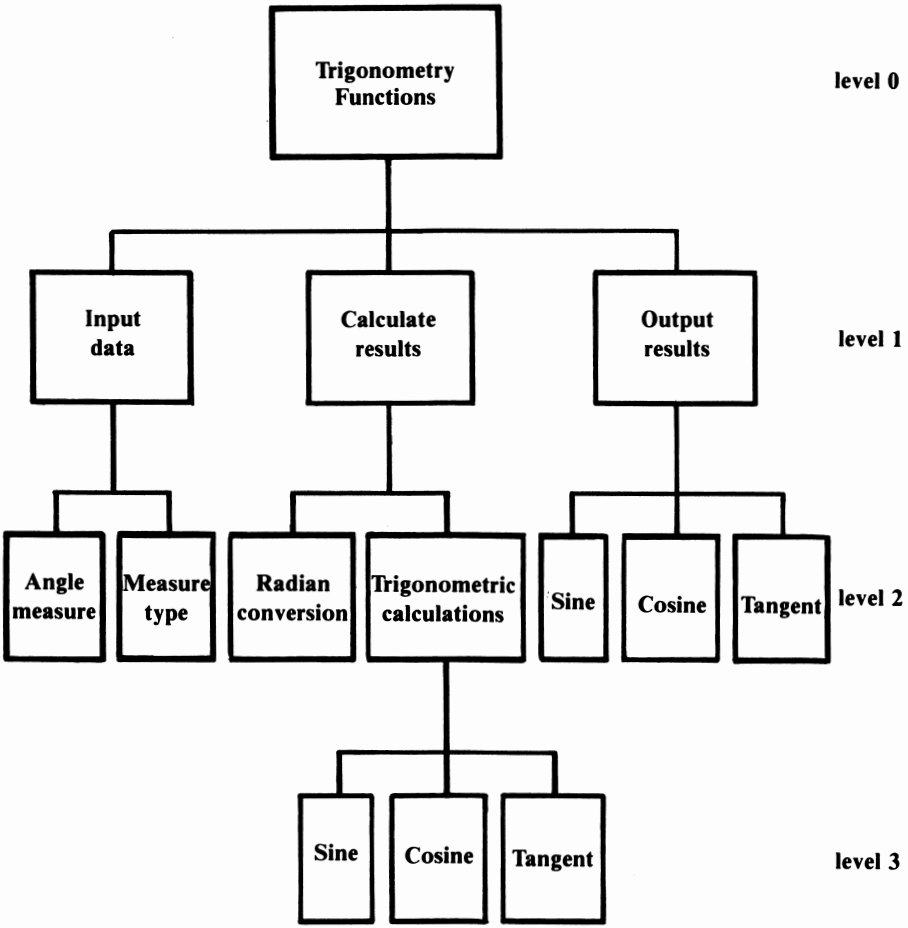
Instead of one large problem, we now have three smaller problems which can be analyzed individually. Our next step is isolate the first level 1 module. We then need to decide whether or not the module, Input data, can be clarified. In order to make this decision, we need to know what data will be input. Obviously, the measure of an angle will be specified. Since that measure may be entered in either degrees or radians, we will also want to know which type was used. These are level two modules. Our diagram now has the following structure:



Next we study the module, Calculate results. The problem definition states that the sine, cosine, and tangent of the angle are to be returned. First, however, we will need to convert the angle's measure to radians if it has been input in degree form. We can group these calculations into two categories: radian conversion and trigonometric calculations. These categories are level two modules. The specific trigonometric functions are level three modules. Our diagram now has the following structure:



The results still have to be output. The results are the sine, cosine, and tangent of the angle. These three components comprise the final level 2 modules:



Our top-down design is now complete. We can begin to write the program. It is a good idea to use the higher level module names in REM statements. If a top-down design is complete, it should be easy to translate into a program. The following program is a sample:

```
Ok
LIST
10 REM***TRIGONOMETRY FUNCTIONS***
20 REM*INPUT DATA*
30 CLS
40 INPUT "Enter the measure of the angle";ANGLE
50 INPUT "Enter the measure type(D/R)";TYPE$
60 REM*CALCULATE RESULTS*
70 REM*CONVERT?
80 IF TYPE$ = "D" THEN ANGLE = ANGLE * .017453
90 REM*TRIGONOMETRIC CALCULATIONS
100 SNANG = SIN(ANGLE)
110 CSANG = COS(ANGLE)
120 TNANG = TAN(ANGLE)
130 REM*OUTPUT RESULTS*
140 PRINT "The sine of the angle =";SNANG
150 PRINT "The cosine of the angle =";CSANG
160 PRINT "The tangent of the angle =";TNANG
170 REM***END***
180 END
Ok
RUN
```

Line 10 uses the name of the level 0 module as a name for the program. The REM statement in line 20 contains the name of the first level 1 module. Line 30 clears the screen. When line 40 is executed, the angle's measure is requested. Line 50 asks whether the measure is in degrees or radians.

The area of the program which corresponds to the second level 1 module begins with line 60. Lines 60 and 70 contain REM statements. In line 80 the angle measure, if specified in degrees, is converted to radians. The level 2 module, Trigonometric calculations, calls three level 3 modules. Line 90 identifies the level 2 module, while lines 100 through 120 calculate the sine, cosine, and tangent of the angle. In those lines the results are also assigned to variables.

Line 130 begins the next level 1 module. It has three output modules at level 2. Lines 140 through 160 output the data. The program ends with lines 170 and 180.

Delay Routine

The purpose of a delay routine is to postpone but not prevent the occurrence of the next event in a sequence. A delay routine may or may not contain an actual loop. The graphics programs in lesson 21 often used a loop such as the following:

```
FOR J = 1 TO 1000:NEXT J
```

The duration of the delay is dependent upon the amount of time which the PC*jr* requires in order to complete the specified number of loops. As the number of loops increases, the duration also increases. This type of delay routine is often used to postpone clearing the screen.

The second type of delay routine does not involve looping. The input of a character or characters is required before the program's execution can continue. The following modification of the previous program demonstrates this type of delay routine:

```
Ok
LIST
10 REM***TRIGONOMETRY FUNCTIONS***
20 REM*INPUT DATA*
30 CLS
40 INPUT "Enter the measure of the angle";ANGLE
50 INPUT "Enter the measure type(D/R)";TYPE$
60 REM*CALCULATE RESULTS*
70 REM*CONVERT?
80 IF TYPE$ = "D" THEN ANGLE = ANGLE * .017453
90 REM*TRIGONOMETRIC CALCULATIONS
100 SNANG = SIN(ANGLE)
110 CSANG = COS(ANGLE)
```

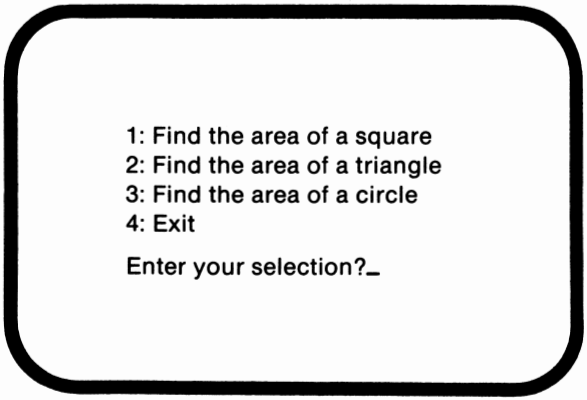
program continued on next page

```
120 TNANG = TAN(ANGLE)
130 REM*OUTPUT RESULTS*
140 PRINT "The sine of the angle =";SNANG
150 PRINT "The cosine of the angle =";CSANG
160 PRINT "The tangent of the angle =";TNANG
170 REM*DELAY ROUTINE*
180 PRINT: PRINT "Press any key to clear screen."
190 K$ = INPUT$(1)
200 REM***END***
210 CLS
220 END
Ok
RUN
```

Lines 10 through 160 are unaltered. Line 170, however, uses a REM statement to identify the addition. Line 180 prints the prompt. When line 190 is executed, the program waits for a key to be pressed. Once this condition is met, lines 200 through 220 are executed. The screen is cleared, and the program ends.

Menu-Driven Programming

The programs which have been presented in this book have been short. Several short, related programs may be grouped into one large program. The different options are then presented as part of a **menu**. A menu in computer programming is similar to a restaurant menu in that it presents a number of available selections. The following diagram illustrates a possible menu screen display:



The person who is running the program is presented with a list of choices such as the one above and is asked to make a selection. That choice is then executed. The following program demonstrates the use of menu-driven programming:

```
Ok
LIST
10 REM***SIMPLE MATH***
20 REM*MENU*
30 REM*PRINT MENU
40 CLS: PRINT "1: Add two numbers"
50 PRINT "2: Subtract two numbers"
60 PRINT "3: Multiply two numbers"
70 PRINT "4: Divide two numbers"
80 REM*INPUT CHOICE
90 PRINT: INPUT "Enter selection";NUM
100 REM*INPUT NUMBERS*
110 PRINT: INPUT "Enter two numbers";NUM1,NUM2
120 REM*SUBROUTINE CALLS*
130 IF NUM = 1 THEN GOSUB 1000
140 IF NUM = 2 THEN GOSUB 2000
```

program continued on next page

```
150 IF NUM = 3 THEN GOSUB 3000
160 IF NUM = 4 THEN GOSUB 4000
170 REM*PRINT RESULTS*
180 PRINT "The result =";ANSWER
190 REM***END***
200 END
1000 REM*ADDITION SUBROUTINE*
1010 ANSWER = NUM1 + NUM2
1020 RETURN
2000 REM*SUBTRACTION SUBROUTINE*
2010 ANSWER = NUM1 - NUM2
2020 RETURN
3000 REM*MULTIPLICATION SUBROUTINE*
3010 ANSWER = NUM1 * NUM2
3020 RETURN
4000 REM*DIVISION SUBROUTINE*
4010 ANSWER = NUM1 / NUM2
4020 RETURN
Ok
RUN
```

Lines 10 through 30 contain REM statements. Lines 40 through 70 result in the menu being printed. Line 80 is a comment. When line 90 is executed, the number of the menu selection is requested. This is followed by lines 100 and 110 which request the two operands. Lines 120 through 160 contain the subroutine calls. If NUM is equal to 1, the addition subroutine in lines 1000 through 1020 is executed. In this subroutine the numbers are summed. On the other hand, if NUM is equal to 2, the subtraction subroutine which begins in line 2000 is executed. The second value is then subtracted from the first. A value of 3 stored in NUM indicates that multiplication is to be performed. The subroutine in lines 3000 through 3020 is then executed. Finally, if NUM is equal to 4, the values are divided by the subroutine which is contained in lines 4000 through 4020.

In line 170 a REM statement indicates a new segment of the program. Line 180 prints the result. Lines 190 and 200 end the program.

Menu-driven programming is, in certain situations, a natural result of using top-down design. Whenever only one of a series of modules on a level is to be executed, a menu may be an excellent choice.

Techniques using Variables

Several simple programming techniques involve variables. In this section of the lesson we will discuss initializing variables, using variables as flags, and using significant variable names.

Initializing Variables

Initializing variables is the process of assigning specific values to certain variables prior to their usage within a program. The need for initializing variables can best be illustrated using a simple analogy. Suppose you were counting the change in each of your pockets. You would normally begin counting at zero and sum the value of the coins. When you begin to count the change in your other pocket, you begin counting at zero again. This process of initializing variables is similar to beginning your count with zero. In programming, initialization is most often used when working with loops. Running the following program will demonstrate how failing to initialize certain variables can cause errors:

```
Ok
LIST
10 REM***MILEAGE***
20 CLS: INPUT "Enter number of cars";CARS
30 FOR LOOP = 1 TO CARS
40 CLS: PRINT TAB(15) "CAR NUMBER";LOOP
50 PRINT: INPUT "Enter number of trips";TRIP
60 FOR LOOP2 = 1 TO TRIP
70 INPUT "Enter miles";MILES
80 DRIVEN = DRIVEN + MILES
90 NEXT LOOP2
100 INPUT "Enter gallons of gasoline";GASOLINE
110 MILEAGE = DRIVEN / GASOLINE
120 PRINT: PRINT "Mileage =";MILEAGE
130 PRINT: PRINT "Press any key to continue."
140 K$ = INPUT$(1)
150 NEXT LOOP
160 REM***END***
170 CLS
180 END
Ok
RUN
```

Line 10 contains a REM statement which identifies the program. Line 20 requests the number of cars. This value is used to determine the length of the loop which begins in line 30. Each time the loop is executed, the screen is cleared and the car number is listed. This occurs in line 40. In line 50 the person who is running the program is asked to enter the number of trips which that car made. Lines 60 through 90 comprise a loop which sums the miles driven on each trip. The number of repetitions of the loop is equal to the number of trips. When line 100 executes, the user is asked to enter the total number of gallons of gasoline which was used on those trips. The mileage is then determined in line 110 by dividing the miles driven by the gallons of gasoline which were used. Line 120 prints the result. Lines 130 and

140 contain the second type of delay routine. Execution to determine the mileage for the second car is postponed until a key is pressed. Then the loop is incremented, and execution continues.

Notice that when this program is run, the results are unreasonable except for the first car. This effect is the result of not initializing the variable DRIVEN. The number of miles driven is added to the number of miles driven by all previous cars. The addition of line 35 corrects the problem:

```
Ok
LIST
10 REM***MILEAGE***
20 CLS: INPUT "Enter number of cars";CARS
30 FOR LOOP = 1 TO CARS
35 DRIVEN = 0
40 CLS: PRINT TAB(15) "CAR NUMBER";LOOP
50 PRINT: INPUT "Enter number of trips";TRIP
60 FOR LOOP2 = 1 TO TRIP
70 INPUT "Enter miles";MILES
80 DRIVEN = DRIVEN + MILES
90 NEXT LOOP2
100 INPUT "Enter gallons of gasoline";GASOLINE
110 MILEAGE = DRIVEN / GASOLINE
120 PRINT: PRINT "Mileage =";MILEAGE
130 PRINT: PRINT "Press any key to continue."
140 K$ = INPUT$(1)
150 NEXT LOOP
160 REM***END***
170 CLS
180 END
Ok
RUN
```

DRIVEN was initialized in line 35. If the initialization had occurred prior to line 30, the value would not have been reset between cars. If initialization had occurred after line 60, the variable would have been reset with every trip, not every car.

Flags

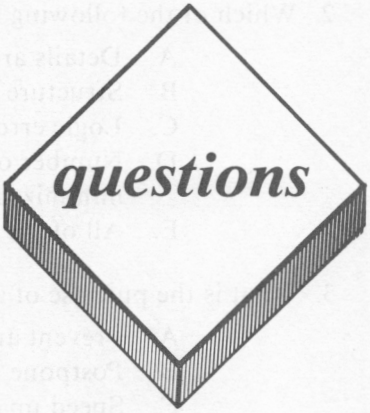
A flag is a variable with either a true or a false value. This value indicates whether or not a condition or process is complete. An everyday example of a flag is a dryer buzzer. When the drying time has run out, the buzzer sounds. In programming, a flag is used most often with an IF THEN controlled loop or with a WHILE, WEND loop. The following program illustrates the use of a flag:

```
Ok
LIST
10 REM***FLAG PROGRAM***
20 CLS
30 INCOMPLETE = -1
40 WHILE INCOMPLETE
50 INPUT "Enter a number(0 to end)";VALUE
60 IF VALUE = 0 THEN INCOMPLETE = 0
70 SUM = SUM + VALUE
80 WEND
90 PRINT "Sum =";SUM
100 END
Ok
RUN
```

Line 10 contains a REM statement which identifies the program. In line 20 the screen is cleared. When line 30 is executed, INCOMPLETE, the flag variable, is assigned a value of true. As long as INCOMPLETE evaluates to true, the loop in lines 40 through 80 will repeat. In that loop, a value is requested. If the value is equal to zero, INCOMPLETE is set to false. The values are summed in line 70. When INCOMPLETE is equal to 0, the sum is printed. The value which is stored in the flag indicates whether or not the process is complete.

Significant Variable Names

By now you are probably aware that it is easier to follow programs in which the variable names have meaning. Using meaningful variable names is called using significant variable names. Programs which employ this technique are easier to debug and require fewer comments.



True or False

1. A delay routine must contain an actual loop.
2. Menu-driven programming and top-down design are incompatible.
3. The process of assigning a variable a specific value prior to the use of that variable is called flagging.
4. Using variable names which indicate the functions of the variables is called using significant variable names.

Multiple Choice

1. What is the first step in programming?
   A. Initializing the variables
   B. Writing the program
   C. Flagging
   D. Creating a top-down design
   E. None of the above

2. Which of the following is a result of top-down design?
   A. Details are deferred
   B. Structure is more easily achieved
   C. Logic errors are reduced
   D. Number of GOTO statements is minimized
   E. All of the above

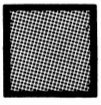
3. What is the purpose of a delay routine?
   A. Prevent an event from occurring
   B. Postpone the occurrence of an event
   C. Speed up execution
   D. Halt execution
   E. None of the above

4. Which of the following groups several short programs together?
   A. Top-down design
   B. Delay routines
   C. Initializing variables
   D. Using significant variable names
   E. None of the above

Essay

1. Create a top-down design which describes how to order a pizza.

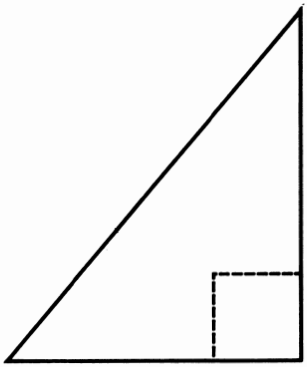
Section 3



In Section 2 you began programming the computer. You gained a general understanding of Microsoft BASIC's history and learned how to operate the PCjr. You also worked with several BASIC commands and statements. Finally, you learned some special programming techniques.

In this section titled "APPLICATIONS", we will present several ways in which you might use the PCjr to aid in your school work. These ideas are not meant to be a complete listing. They are merely intended as a starting point for your own ideas. In this section we will attempt to accomplish the following goals:

- ☐ *Explore how the PCjr can contribute to the study of mathematics*
- ☐ *Discuss ways in which the PCjr can be applied to the field of science*
- ☐ *Explain how to use the PCjr with a word processing program to format papers and reports*



Applications for Mathematics

lesson 24

Lesson Goals

- *Consider ways in which the PCjr can be used to solve problems involving algebra*
- *Explore methods in which the PCjr can be used to work with geometry*
- *Examine ways in which the PCjr can assist in the study of trigonometry*

Introduction

Now that you have learned the elementary concepts of BASIC, you can begin to apply that knowledge. By this time you are probably familiar with the PC*jr*'s entertainment value. The PC*jr*, however, can also be used to transform tedious tasks into more enjoyable experiences. In this lesson we will discuss some ways you can apply your programming skills in conjunction with the PC*jr* to increase the effectiveness of the time you spend studying mathematics.

Algebra

Writing programs to perform computations can be very efficient. This is especially true if the computations involved will be used repeatedly. Writing a program may also be the most efficient means of processing large amounts of data. Even when it is not the most efficient option, writing a program to solve a problem which involves a method which you do not completely understand may be an excellent idea. In the process of writing the program you may well find that you have mastered the method.

The PC*jr* can be programmed to solve a number of different types of problems, including algebraic ones. We do not intend, however, to present all of the possibilities. In this section we will present and discuss one example in detail. Programming is esentially a creative process, so the examples in this section and the ones following are designed to serve as a springboard for your own ideas.

One type of problem which the PC*jr* can solve, given the right program, is finding the roots of a quadratic equation. Remember that a quadratic equation has the form:

$$ax^2 + bx + c$$

where a and b are coefficients, and c is a constant. The following program uses the quadratic formula, $(-b \pm (b^2 - 4ac)^{1/2}) / (2a)$, to find the roots of any quadratic equation:

```
Ok
LIST
10 REM***QUADRATIC FORMULA***
20 CLS
30 INPUT "Enter the X   2 coefficient";A
40 INPUT "Enter the X coefficient";B
50 INPUT "Enter the constant";C
60 D = (B ^ 2) - (4 * A * C)
70 IF D < 0 THEN GOTO 140
80 X1 = (-B + SQR(D)) / (2 * A)
90 X2 = (-B - SQR(D)) / (2 * A)
100 IF D = 0 THEN GOTO 180
110 PRINT "X = ";X1;" or X =";X2
120 INPUT "Do you wish to continue(Y/N)";ANS$
130 IF ANS$ = "Y" THEN GOTO 20 ELSE GOTO 200
140 R = -B / (2 * A)
150 I = (ABS(D) ^ .5) / (2 * A)
160 PRINT "X = ";R;"+";I;"i or X = ";R;" - ";I;"i"
170 GOTO 120
180 PRINT "X = ";X1
190 GOTO 120
200 CLS
210 END
Ok
RUN
```

The program begins with a REM statement. The REM statement contains the title of the program. After the screen is cleared in line 20, the person running the program is asked to input the value of the coefficients and the constant.

The computations begin when line 60 is executed. In line 60 the discriminant's value is determined. The discriminant, D, is b^2 - 4ac and is located under the square root sign. If the discriminant's value is negative, the roots are complex, and execution branches to line 140. As long as the value of the discriminant is positive or zero, the equation has at least one rational root. These roots are determined in lines 80 and 90 by applying the quadratic formula. The discriminant is not recalculated since its value has not changed.

If the value of D is zero, the equation has only one root. In this case, the statement in line 100 causes the program to branch to line 180. Line 110 contains the output routine for two rational roots.

In lines 120 and 130, the person running the program is asked to decide whether or not the program is to be executed again. If you intend to work with more than one set of data at a sitting, it is a good idea to include a routine in which the user is asked to make such a choice. Line 140 is never executed if the roots are rational. If the roots are complex, however, their rational part is determined by the computation contained in line 140. In constrast, line 150 calculates the irrational part of the roots. Lines 170 and 190 branch to line 120. Line 180 is only executed if both roots are the same. Lines 200 and 210 clear the screen and end the program.

Geometry

The PC*jr* can also be programmed to solve problems which involve geometry. The calculations which are required to solve geometry problems are seldom complicated but often plentiful. Programs to handle this type of a situation may be structured in a variety of ways. One option, as we discussed in the last lesson, is to write a short program for each calculation. For example, you could write one program to find the area of a square and another to find the area of a triangle. Suppose, however, you need to find the area of a figure which includes both a square portion and a triangular portion. In the process of obtaining the area of the triangular portion with the second program, the area of the square portion, which was returned by the first program, will be lost. These difficulties can be avoided by grouping several small programs into one larger menu-driven program. The data can then be more easily shared. The following program is menu-driven and can calculate the area of a triangle, the length of a side, or the measure of an angle of the triangle:

```
Ok
LIST
10 REM***TRIANGLE PROGRAM***
20 CLS: PRINT TAB(18) "MENU"
30 PRINT: PRINT "1: Area of a triangle"
40 PRINT "2: Length of third side(2 sides given)"
50 PRINT "3: Measure of third angle(2 angles given)"
60 PRINT "4: Exit":PRINT
70 INPUT "Enter the number of the desired option";NUM
80 IF NUM = 1 THEN GOTO 200
90 IF NUM = 2 THEN GOTO 300
100 IF NUM = 3 THEN GOTO 500
110 IF NUM = 4 THEN GOTO 600
120 PRINT "Invalid selection.":PRINT
130 GOTO 70
200 REM*AREA
210 CLS: INPUT "Enter height";H
220 INPUT "Enter base";B
230 A = .5 * (B * H)
240 PRINT "The area is";A
250 GOSUB 1000
260 GOTO 20
300 REM*LENGTH OF SIDE
310 CLS:INPUT "Enter length of leg";A
320 INPUT "Enter length of other leg(0 if unknown)";B
330 INPUT "Enter length of hypotenuse(0 if unknown)";C
340 IF C = 0 THEN GOTO 380
350 B = SQR (C ^ 2 - A ^ 2)
360 PRINT "The length of the leg is";B
370 GOTO 400
380 C = SQR (A ^ 2 + B ^ 2)
390 PRINT "The length of the hypotenuse is";C
400 GOSUB 1000
410 GOTO 20
500 REM*THIRD ANGLE
510 CLS: INPUT "Enter first angle";ANG1
520 INPUT "Enter second angle";ANG2
530 ANG3 = 180 - (ANG1 + ANG2)
540 PRINT "The third angle is";ANG3
550 GOSUB 1000
560 GOTO 20
```

program continued on next page

```
600 REM*EXIT
610 CLS
620 END
1000 REM*DELAY SUBROUTINE*
1010 PRINT "Press any key to continue."
1020 A$ = INPUT$(1)
1030 RETURN
Ok
Run
```

The REM statement in line 10 assigns a name to the program. Lines 20 through 60 list the menu selections. Note that the option specified in line 60 allows the program to be exited gracefully. A selection similar to line 60 is frequently included as part of a menu-driven program. In line 70 the person running the program is asked to enter the number of the selected option. This number determines which of the subprograms will be executed. Lines 80 through 110 branch to the appropriate subprogram. If the value which is entered is not one of the menu options, the message, "Invalid selection," appears on the screen, and the user is asked to make another choice. This process is defined in lines 120 and 130.

Lines 200 through 260 comprise the subroutine which calculates the area of a triangle. Notice that the subprogram is identified by a REM statement. When lines 210 and 220 execute, the person running the program is asked to enter the values for the height and base. These values are multiplied together and divided by two when line 230 executes. The result of this computation, the area of a triangle, is printed by line 240. In line 250 a subroutine is called. This subroutine is a delay subroutine which waits to clear the screen until a key is pressed. After the subroutine has executed, the statement in line 260 branches to the menu portion of the program.

Lines 300 to 410 comprise the subprogram which is used to determine the length of the third side of a triangle. To make the subprogram easier to find, line 300 contains an identifying REM statement. Lines 310 through 330 ask the person who is running the program to enter the lengths, using a 0 to identify the missing side. If the hypotenuse's length needs to be determined, lines 350 through 370 are not executed due to the effect of line 340. If the leg's length is missing, the length of the specified leg is squared and subtracted from the square of the hypotenuse. The square root of this result is determined and assigned to B. This process, which is defined in line 350, is based upon the Pythagorean Theorem. The Pythagorean Theorem states that the square of the length of the hypotenuse is equal to the sum of the squares of the legs' length.

Line 360 prints the value of B along with an appropriate message. Line 370 branches to line 400. If the length of the hypotenuse is unknown, A and B are squared and summed. The square root of the resultant value is calculated and assigned to C in line 380. Line 390 prints the results. After the results are printed, line 400 branches to the delay subroutine. Once the subroutine has been executed, the menu appears.

Lines 500 through 560 are used to determine the measure of a missing angle. In lines 510 and 520, the user is asked to input the values for the two known angles. Since the sum of the degree measures of the interior angles of the triangle must, by definition, equal 180 degrees, the values of the known angles are subtracted from 180, and the result is assigned to ANG3 in line 530. Line 540 prints the result. Lines 550 and 560 execute the delay subroutine and then branch to the menu, respectively.

In lines 600 through 620, the screen is cleared, and the program is ended. Lines 1000 through 1030 comprise the delay subroutine. This subroutine causes the program to wait for the user to indicate that the screen can be cleared.

Another program structure is often desired when several distinct calculations are to be performed with the same items of data. The following program computes the distance, midpoint, and slope of a line which connects two points:

```
Ok
LIST
10 REM***DISTANCE, MIDPOINT, SLOPE***
20 CLS:INPUT "Enter X coordinate of first point";X1
30 INPUT "Enter Y coordinate of first point";Y1
40 INPUT "Enter X coordinate of second point";X2
50 INPUT "Enter Y coordinate of second point";Y2
60 PRINT
70 XDIS = X2 - X1
80 YDIS = Y2 - Y1
90 DIST = SQR(XDIS ^ 2 + YDIS ^ 2)
100 IF XDIS = 0 OR YDIS = 0 THEN GOTO 120
110 M = YDIS / XDIS
120 XMID = (X2 + X1) / 2
130 YMID = (Y2 + Y1) / 2
140 PRINT "The distance is";DIST
150 PRINT "The midpoint is (";XMID;",";YMID;")"
160 IF XDIS = 0 THEN PRINT "The line is vertical.":GOTO 190
170 IF YDIS = 0 THEN PRINT "The line is horizontal.":GOTO 190
180 PRINT "The slope is";M:PRINT
190 INPUT "Do you wish to continue(Y/N)";ANS$
200 IF ANS$ = "Y" THEN 20
210 CLS
220 END
Ok
Run
```

Lines 10 through 60 serve as the introduction to the program. Line 10 names the program. Lines 20 through 50 allow the user to input the coordinates of the line segment's end points. These coordinates are stored in the variables X1, Y1, X2, and Y2.

In lines 70 and 80, preliminary computations are performed. Line 70 calculates the distance between the two X coordinates. Line 80 calculates the distance between the Y coordinates. These values, the coordinate-specific distances, are needed for the calculation which line 90 defines. This calculation is based upon the distance formula which is as follows:

$$\text{Distance between two points} = ((X2 - X1)^2 + (Y2 - Y1)^2)^{1/2}$$

Lines 100 and 110 determine the slope of the line. If either XDIS or YDIS is equal to zero, line 110 is skipped. The calculation of the slope in line 110 is based upon the following slope formula:

Slope = (Y2 - Y1) / (X2 - X1)

This formula is often described as "rise over run."

The final set of calculations determines the coordinates of the line's midpoint. The X coordinate of the midpoint is computed in line 120 and is equal to the sum of the X coordinate values divided by two. The same basic equation is used in line 130 to calculate the Y coordinate of the midpoint.

Lines 140 through 180 print the results. The distance and midpoint are displayed on the screen as a result of the statements in lines 140 and 150. If the line is vertical, that fact is conveyed by line 160. Line 170's message is only displayed if the line is horizontal. Line 180 is executed only if neither of the previous conditions is met. Lines 190 through 210 contain the ending routine.

Trigonometry

The calculations required to solve trigonometry problems are often complex. In this section we will present one program which combines many of the factors we discussed in lesson 23, including a reliance upon subroutines for structure. The following program is designed to use trigonometry to solve for the remaining dimensions of a right triangle when given two dimensions, at least one of which is the length of a side:

```
Ok
LIST
10 REM***TRIG PROGRAM***
20 CLS
30 X = 0:Y = 0:DEG = 0:DEG2 = 0:HYP = 0:ADJ = 0:OPP =0:DONE = 0
40 PRINT "PARTS OF A TRIANGLE"
50 PRINT:PRINT "1: Angle in degrees"
60 PRINT "2: Length of adjacent side"
70 PRINT "3: Length of opposite side"
80 PRINT "4: Length of hypotenuse"
90 PRINT:INPUT "Which two parts are known";X,Y
100 IF X = 1 OR Y = 1 THEN INPUT "Enter angle";DEG
110 IF X = 2 OR Y = 2 THEN INPUT "Enter length of adjacent side";ADJ
120 IF X = 3 OR Y = 3 THEN INPUT "Enter length of opposite side";OPP
130 IF X = 4 OR Y = 4 THEN INPUT "Enter length of hypotenuse";HYP
140 IF DEG <> 0 THEN GOSUB 1000
150 IF DONE THEN GOTO 210
160 IF ADJ <> 0 AND OPP <> 0 THEN GOSUB 2000
170 IF DONE THEN GOTO 210
180 IF ADJ <> 0 AND HYP <> 0 THEN GOSUB 3000
190 IF DONE THEN GOTO 210
200 IF OPP <> 0 AND HYP <> 0 THE GOSUB 4000
210 PRINT "The angle =";DEG
220 PRINT "The other angle =";DEG2
230 PRINT "The adjacent side =";ADJ
240 PRINT "The opposite side =";OPP
250 PRINT "The hypotenuse =";HYP
260 PRINT:INPUT "Do you wish to continue(Y/N)";ANS$
270 IF ANS$ = "Y" THEN GOTO 20
280 CLS
290 END
1000 REM*ANGLE SUBROUTINE*
1010 DEGR = DEG * .017453
1020 DEG2 = 90 - DEG
1030 IF ADJ <> 0 THEN GOTO 1090
1040 IF HYP <> 0 THEN GOTO 1130
1050 REM*KNOW OPP,DEG
1060 HYP = OPP * (1 / SIN(DEGR))
1070 ADJ = OPP * (1 / TAN(DEGR))
1080 DONE = -1:RETURN
1090 REM*KNOW ADJ,DEG
1100 OPP = ADJ * TAN(DEGR)
1110 HYP = ADJ * (1 / COS(DEGR)
```

program continued on next page

```
1120 DONE = -1:RETURN
1130 REM*KNOW HYP,DEG
1140 ADJ = HYP * COS(DEGR)
1150 OPP = HYP * SIN(DEGR)
1160 DONE = -1:RETURN
2000 REM*KNOW ADJ,OPP SUBROUTINE*
2010 DEG = ATN (OPP / ADJ) * 57.29578
2020 HYP = SQR (ADJ ^ 2 + OPP ^ 2)
2030 DEG2 = 90 - DEG
2040 DONE = -1:RETURN
3000 REM*KNOW ADJ,HYP SUBROUTINE*
3010 OPP = SQR (HYP ^ 2 - ADJ ^ 2)
3020 DEG = ATN (OPP / ADJ) * 57.29578
3030 DEG2 = 90 - DEG
3040 DONE = -1:RETURN
4000 REM*KNOW OPP,HYP SUBROUTINE*
4010 ADJ = SQR (HYP ^ 2 - OPP ^ 2)
4020 DEG = ATN (OPP / ADJ) * 57.29578
4030 DEG2 = 90 - DEG
4040 DONE = -1:RETURN
Ok
Run
```

Lines 10 through 90 introduce the program. Line 10 provides a name for the program. After the screen is cleared in line 20, several variables are initialized in line 30. Because of the program's structure, these variables need to be reset to 0 before each execution. Lines 40 through 80 list the menu. Line 90 asks the person running the program to identify which two parts of the triangle have known dimensions. That data is stored in X and Y.

Lines 100 through 130 comprise the input routine. If the user has indicated that the measure of an angle is known, that information is requested by line 100. The measure of the angle is then assigned to DEG. If either X or Y is equal to two, the length of the adjacent side, ADJ, is requested. This process occurs in line 110. In line 120 the length of the opposite side is requested if X or Y is equal to 3. OPP then stores that value. Finally, if X or Y is equal to 4, the person running the program is asked to enter the length of the hypotenuse. HYP is then assigned that value.

Another complicated series of GOSUB and GOTO statements appears in lines 140 through 200. Remember that the values of all the variables used in this segment of the program are equal to zero, except for those values which the user input. Line 140 checks the value stored in DEG. If that value is not equal to zero, then the person running the program specified the measure of an angle. In this case, the subroutine which calculates the dimensions of the remaining sides and angle is called. This subroutine begins at line 1000.

Within the subroutine, DONE is set to true, or -1. DONE is a flag which indicates that the dimensions of all of the sides and angles are known. Line 150 checks the status of DONE. If DONE is equal to -1, the program branches to line 210.

If the dimensions are not yet calculated, line 160 is executed. The condition in line 160 evaluates to true only if both ADJ and OPP have been assigned nonzero values. If this is true, the subroutine which begins at line 2000 is executed. Within this subroutine, DONE is also assigned a value of -1. Line 170 contains another check on DONE's status. If DONE is false, no branching occurs.

In contrast, line 180 checks whether ADJ and HYP have nonzero values. If this condition evaluates to true, the subroutine in lines 3000-3040 is called. Again, DONE is set to true within the subroutine, and line 190 contains a check on the flag's status. If DONE is still equal to zero, line 200 is executed. Line 200 calls the subroutine which begins in line 4000 if the values for OPP and HYP do not equal zero.

To make this discussion clearer, we will trace an example. Suppose the values of ADJ and HYP have been specified. The condition in line 140 evaluates to false because DEG is still equal to zero. The GOSUB 1000 statement is not executed. Since DONE's value is unchanged, that condition evaluates to false, and line 160 is executed. The value of ADJ does not equal 0, but the value of OPP is still equal to zero. The condition thus evaluates to false. DONE's value remains equal to zero, so line 180 is executed after line 170. Since both ADJ and HYP contain nonzero values, the subroutine which begins at line 3000 is executed, and the remaining dimensions are calculated. DONE is set to true before RETURN is encountered. When line 190 is executed, the condition evaluates to true, and the program branches to line 210. Line 200 is never executed.

Lines 210 through 250 print the results of the calculations. Appropriate messages are also output. Lines 260 through 290 comprise the routines which ask whether the program should continue. Unless the answer is yes, the program will end.

The longest subroutine is contained in lines 1000 through 1160. This subroutine is composed of four smaller areas. The first of these areas contains the calculations which are needed by the subsequent areas and consists of lines 1010 through 1040. In line 1010 the value in DEG is converted to its equivalent in radians. The resultant value is assigned to DEGR. In line 1020 the value for the other angle is computed. Since the triangle is a right angle, by definition, one angle is a right angle which measures 90 degrees. Ninety subtracted from 180 leaves 90 degrees which are shared by the other two angles. Since DEG has been specified, DEG2 is equal to DEG subtracted from 90. DEG2 is the other angle. Lines 1030 and 1040 determine the branching. If the value of ADJ is nonzero, the program branches to line 1090. If HYP has a nonzero value, line 1040 branches to line 1130.

If neither of these conditions evaluates to true, lines 1050 through 1080 are executed. HYP is calculated in line 1060 by taking the reciprocal of the sine of DEGR and multiplying that value by the value stored in OPP. Remember that the sine of an angle is equal to the length of the opposite side divided by the length of the hypotenuse. In order for this value multiplied by the length of the opposite side to yield the length of the hypotenuse, we need a slightly different equation. The following equation will suffice:

$$\text{length of hypotenuse} = \text{length of opposite side} * \frac{\text{hypotenuse}}{\text{opposite side}}$$

The hypotenuse divided by the opposite side is the definition of the cosecant. The PC*jr*, however, does not have a cosecant function. The cosecant, though, is the reciprocal of the sine and the PC*jr* does have a built-in sine function. Table 24.1 lists the trigonometric functions and their interrelationships. The cosecant function can be simulated by dividing one by the sine of DEG. The result is assigned to HYP.

Table 24.1. Trigonometric Functions

Name	Definition	Relationship
sin A =	$\frac{\text{opposite}}{\text{hypotenuse}}$	
cosA =	$\frac{\text{adjacent}}{\text{hypotenuse}}$	
tanA =	$\frac{\text{opposite}}{\text{adjacent}}$	$\frac{\text{sine A}}{\text{cos A}}$
sec A =	$\frac{\text{hypotenuse}}{\text{adjacent}}$	$\frac{1}{\text{cos A}}$
csc A =	$\frac{\text{hypotenuse}}{\text{opposite}}$	$\frac{1}{\text{sin A}}$
cot A =	$\frac{\text{adjacent}}{\text{opposite}}$	$\frac{1}{\text{tan A}}$

Once two sides are known, the third side's length can be determined using the Pythagorean Theorem. In this case, however, we chose to simulate the cotangent function. The cotangent function is equal to the reciprocal of the tangent. Remember that the tangent of a value is equal to the length of the opposite side divided by the length of the adjacent side. The reciprocal, the cotangent of a value, is equal to the length of the adjacent side divided by the length of the opposite side. When this value is multiplied by OPP, ADJ, the adjacent side's length, is returned. This calculation occurs in line 1070. In line 1080 DONE is assigned the value -1, and the subroutine is exited.

Lines 1090 through 1120 comprise the area of the subroutine which is executed if ADJ's value is known. Line 1090 identifies the region with a REM statement. In line 1100 the length of the hypotenuse is calculated. The secant function is simulated by taking the reciprocal of the cosine of DEGR. As we discussed earlier, the cosine is equal to the length of the adjacent side divided by the length of the hypotenuse. The secant of a value is equal to the length of the hypotenuse divided by the length of the adjacent side. When the

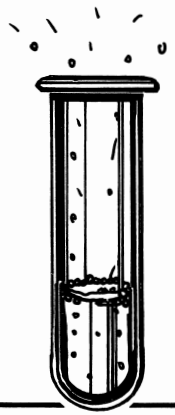
reciprocal of the cosine is multiplied by the length of the adjacent side, the length of the hypotenuse is returned. In contrast, the opposite side's measure can be determined simply by multiplying TAN(DEGR) by ADJ, since the tangent of DEGR is equal to OPP divided by ADJ. This occurs in line 1110. Line 1120 is identical to line 1080. The flag is set to -1, and the subroutine is exited.

Lines 1130 through 1160 are structured similarly to the lines in the two preceding areas. The hypotenuse is the known side. ADJ is equal to HYP multiplied by the cosine of DEGR, since the cosine is the adjacent side's length divided by the sine of DEGR, or opposite side divided by hypotenuse. In line 1160, DONE is assigned the value of -1, and the subroutine is exited.

Line 2000 begins a new subroutine. This subroutine, as the REM statement indicates, is executed if ADJ and OPP have nonzero values. In line 2010 the value of DEG is determined. Remember that OPP divided by ADJ is equal to the tangent of DEG. The arctangent returns the radian measure of an angle whose tangent is equal to the calling value. Since that condition is fulfilled by dividing OPP by ADJ, the arctangent of that value returns the radian measure of DEG. Prior to storing the value, it is converted to degrees. In line 2020 the length of the hypotenuse is calculated by using the Pythagorean Theorem. Line 2030 computes the value of DEG2, and line 2040 sets the flag and exits the subroutine.

The subroutines which begin at lines 3000 and 4000 are very similar to lines 2000 through 2040. In each, the length of the missing side is calculated using the Pythagorean Theorem. DEG and DEG2's values are calculated using exactly the same process as in lines 2010 and 2030. The last line of each subroutine is also identical to line 2040.

Notice that the use of subroutines divides the program into smaller, more manageable pieces. Each subroutine can be individually written and debugged.



$$H_2 + 1/2\ O_2 = H_2O$$

Applications for Science

lesson 25

Lesson Goals

- *Gain an understanding of how the PCjr can assist in the study of chemistry*
- *Explore the potential for using the PCjr as an aid in the study of physics*

Introduction

In the last lesson, we examined how the PC*jr* can be used in your study of mathematics. The PC*jr* can also assist you in your science studies. In this lesson we will discuss examples of the ways in which the PC*jr* can be useful in your studies in the fields of chemistry and physics.

Chemistry

Solving a chemistry problem may require a number of calculations. Any or all of these calculations can be performed on the PC*jr*, given the right program. In the following sections, we will explore four programs which demonstrate applicable methods.

Conversion Program

Often one of the first steps in obtaining the solution to a chemistry problem is to convert the units with which you were provided into those units with which you want to work. An example is a problem in which the data is specified in inches but the equation requires data in centimeters. Converting the data is simple but tedious. The following program uses a menu and handles a number of standard conversions into the metric system:

```
Ok
LIST
10 REM***CONVERSION PROGRAM***
20 CLS: PRINT TAB(18) "MENU"
30 PRINT: PRINT TAB(11) "Metric Conversions"
40 PRINT: PRINT TAB(11) "1: Length"
50 PRINT TAB(11) "2: Mass"
60 PRINT TAB(11) "3: Temperature"
70 PRINT TAB(11) "4: Volume"
80 PRINT TAB(11) "5: Exit"
90 PRINT: INPUT "Enter your selection";NUM
100 IF NUM = 1 THEN GOTO 170
110 IF NUM = 2 THEN GOTO 270
120 IF NUM = 3 THEN GOTO 370
130 IF NUM = 4 THEN GOTO 470
140 IF NUM = 5 THEN GOTO 570
150 PRINT "Invalid selection."
160 PRINT:GOTO 90
170 REM*LENGTH
180 CLS:PRINT
190 PRINT TAB(17) "LENGTH"
200 PRINT: INPUT "Enter the number of inches";INCH
210 PRINT: CENTI = 2.54 * INCH
220 METER = CENTI / 100
230 PRINT "Centimeters =";CENTI
240 PRINT "Meters =";METER
250 GOSUB 1000
260 GOTO 20
270 REM*MASS
280 CLS:PRINT
290 PRINT TAB(18) "MASS"
300 PRINT: INPUT "Enter number of pounds";LB
310 PRINT: KILO = LB / 2.204
320 GRAM = LB * 453.6
330 PRINT "Kilograms =";KILO
340 PRINT "Grams =";GRAM
350 GOSUB 1000
360 GOTO 20
370 REM*TEMPERATURE
380 CLS: PRINT
390 PRINT TAB(12) "TEMPERATURE"
400 PRINT: INPUT "Enter degrees Farenheit";F
410 PRINT: C = (5 / 9) * (F - 32)
420 K = C + 273.15
```

program continued on next page

```
430 PRINT "Degrees Celsuis =";C
440 PRINT "Degrees Kelvin =";K
450 GOSUB 1000
460 GOTO 20
470 REM*VOLUME
480 CLS: PRINT
490 PRINT TAB(17) "VOLUME"
500 PRINT: INPUT "Enter number of ounces";OZ
510 PRINT: MILLI = OZ * 29.57
520 LITER = MILLI / 1000
530 PRINT "Milliliters =";MILLI
540 PRINT "Liters =";LITER
550 GOSUB 1000
560 GOTO 20
570 REM***END***
580 CLS
590 END
1000 REM*DELAY SUBROUTINE*
1010 PRINT: PRINT "Press any key to continue."
1020 A$ = INPUT$(1)
1030 RETURN
Ok
RUN
```

Lines 10 through 160 comprise the menu portion of the program. A REM statement which identifies the program is contained in line 10. When lines 20 through 80 execute, the menu options appear on the screen. The person who is running the program is asked in line 90 to select one of the options. Lines 100 through 140 then branch based upon the value of NUM to the appropriate area of the program. If NUM is equal to 1, lines 170 through 260 will be executed next. A value of 2 stored in NUM indicates that the region of the program which begins in line 270 should be executed. A value of 3 indicates a branch to line 370, and a value of 4 represents a branch to line 470. If NUM is equal to 5, the program will be exited as a result of the statements which are contained in lines 570, 580, and 590.

Lines 150 and 160 will only be executed if NUM does not store a value of 1, 2, 3, 4, or 5. In this event, a message which indicates that an invalid selection was entered appears on the screen. A new selection is then requested.

The conversion of a number of inches into equivalent metric values is accomplished by lines 170 through 260. In line 170 this area of the program is identified by a REM statement. Line 180 clears the screen, and line 190 is responsible for causing the word, length, to appear on the screen. In line 200 the person who is running the program is asked to input the number of inches. The value is then assigned to INCH. Since 1 inch is equal to 2.54 centimeters, the value of INCH is multiplied by 2.54, and the result is assigned to CENTI. This process occurs in line 210. One meter is equal to 100 centimeters, so the value assigned to METER in line 220 is CENTI divided by 100. In lines 230 and 240, the results are output. Line 250 calls the delay subroutine, and line 260 branches to the menu.

The conversion of mass units occurs in lines 270 through 360. The structure of this part of the program is similar to the structure of the length subprogram. Lines 270 through 290 identify this region of the program. In line 300 the number of pounds is requested and assigned to LB. One kilogram is equal to 2.204 pounds, so the value stored in LB is divided by 2.204 and assigned to KILO. When line 320 executes, LB is multiplied by 453.6 because 453.6 grams are the equivalent of 1 pound. Lines 330 and 340 output the results, and lines 350 and 360 branch to other areas of the program.

The temperature conversions which result from the statements in lines 370 through 460 and the volume conversions contained in lines 470 through 560 share the previous structure. Only the conversion factors, messages, and variable names differ.

Lines 570 through 590 end the program. Line 570 contains a REM statement, and line 580 is responsible for clearing the screen. When line 590 is executed, the program ends.

The delay subroutine is contained in lines 1000 through 1030. The program's execution pauses until a key is pressed.

Percentage Composition

Another required step in the solution of many chemistry problems involves working with grams, moles, and percentage composition. The percentage composition is the percentage of an element by weight in a compound. For example, oxygen is 89% by weight of water, H_20. The following comparatively short program demonstrates this method:

```
Ok
LIST
10 REM***MOLES,GRAMS,PERCENTAGES***
20 CLS: INPUT "Enter number of grams of A";GRMA
30 INPUT "Enter atomic weight of A";ATMA
40 INPUT "Enter atomic weight of B";ATMB
50 INPUT "Enter A:B mole ratio";RATA,RATB
60 INPUT "Enter number of grams of compound";GRMC
70 MOLA = GRMA / ATMA
80 MOLB = (MOLA / RATA) * RATB
90 GRMB = MOLB * ATMB
100 PERCENTA = GRMA / GRMC * 100
110 PERCENTB = GRMB / GRMC * 100
120 PRINT: PRINT "Moles of A =";MOLA
130 PRINT "Moles of B =";MOLB
140 PRINT "Grams of B =";GRMB
150 PRINT "Percentage of A =";PERCENTA
160 PRINT "Percentage of B =";PERCENTB
170 PRINT: INPUT "Do you wish to continue(Y/N)";ANS$
180 IF ANS$ = "Y" THEN GOTO 20
190 CLS
200 END
Ok
RUN
```

Line 10 contains a REM statement which identifies the program. Lines 20 through 60 ask for the necessary data: the grams of A, atomic weight of A, atomic weight of B, the coefficients of A and B in a balanced chemical equation, and the grams of the compound.

Lines 70 through 110 perform the computations. The number of moles of A, MOLA, is determined by dividing the atomic weight of A by the number of grams of A. This process occurs in line 70. Since the number of grams of B is not specified, another method must be used to determine the number of moles of B. In line 80 MOLA is divided by RATA, A's coefficient. In the same line, this result is multiplied by RATB, B's coefficient, yielding the moles of B, MOLB. The grams of B, GRMB, are then determined in line 90 by multiplying MOLB by ATMB. The percentages by weight of A and B are calculated in lines 100 and 110 by dividing the grams of each element by the total weight of the compound.

Lines 120 through 160 comprise the output routine. The person who is running the program is then asked in line 170 whether or not the program should be executed again. If the value entered is "Y", the program branches to line 20 as a result of line 180. Any other answer executes the ending routine in lines 190 and 200.

Limiting Reagent

During a reaction, the reactants combine to form the product or products. The reactants are rarely present in exactly the right proportion. When this is the case, one of the reactants acts as a limit to the quantity of the product which will result. This reactant is the limiting reagent. The following program determines which reactant is the limiting reagent and returns other relevant information:

```
Ok
LIST
10 REM***LIMITING REAGENT***
20 CLS: INPUT "Enter number of grams of A";GRMA
30 INPUT "Enter number of grams of B";GRMB
40 INPUT "Enter atomic weight of A";ATMA
50 INPUT "Enter atomic weight of B";ATMB
60 INPUT "Enter A:B mole ratio";RTAB,RTBA
70 INPUT "Enter A:C mole ratio";RTAC,RTCA
80 INPUT "Enter molecular weight of the product";MWTP
90 MOLA = GRMA / ATMA
100 MOLB = GRMB / ATMB
110 LRCALC = (MOLB / RTBA) - (MOLA / RTAB)
120 IF LRCALC > 0 THEN A = -1 ELSE A = 0
130 PRINT
140 IF LRCALC = 0 THEN PRINT "Both reactants are consumed.":GOTO 260
150 IF A THEN GOTO 220
160 REM*B IS LIMITING REAGENT
170 PRINT "B is the limiting reagent."
180 AMTU = (MOLA - (MOLB * (RTAB / RTBA))) * ATMA
190 PRINT "Grams of A unreacted =";AMTU
200 MOLC = MOLB * (RTCA / (RTAC / RTAB * RTBA))
210 GOTO 270
220 REM*A IS LIMITING REAGENT
230 PRINT "A is the limiting reagent."
240 AMTU = (MOLB - (MOLA * (RTBA / RTAB))) * ATMB
250 PRINT "Grams of B unreacted =";AMTU
260 MOLC = MOLA * (RTCA / RTAC)
270 REM*CLOSING
280 GRMC = MOLC * MWTP
290 PRINT "Moles of product produced=";MOLC
300 PRINT "Grams of product produced  =";GRMC
310 PRINT: INPUT "Do you wish to continue(Y/N)";ANS$
320 IF ANS$ = "Y" THEN GOTO 20
330 CLS
340 END
Ok
RUN
```

Line 10 contains a REM statement which identifies the program. Lines 20 through 80 request information: the grams of A and B, the atomic weights of A and B, the coefficients of A, B, and C, and the molecular weight of C.

The preliminary calculations occur when lines 90 through 150 execute. The number of moles of A, MOLA, is calculated in line 90 by dividing the grams of A, GRMA, by the atomic weight of A which is stored in ATMA. Line 100 calculates the number of moles of B, MOLB, in a similar manner. The calculation which is used to determine whether A or B is the limiting reagent occurs in line 110. MOLB is divided by B's coefficient, RTBA. MOLA is then divided by RTAB, A's coefficient. The second value is subtracted from the first, and the result is assigned to LRCALC. If LRCALC's value is greater than 0, A is the limiting reagent. In line 120 the variable A is assigned a true value if the reactant A is the limiting reagent. Otherwise, A is assigned a value of false, 0. Line 140 checks whether LRCALC is equal to 0, indicating that both reactants were completely consumed. If LRCALC is equal to zero, an appropriate message is printed, and the program branches to line 260. If A is the limiting reagent, line 220 is executed as a result of the statement in line 150.

Lines 160 through 210 are executed only if the reactant B is the limiting reagent. Line 160 contains a REM statement which introduces the section, and line 170 prints an appropriate message. In line 180 the amount of A which remains is calculated. RTAB is divided by RTBA. This result is multiplied by MOLB and then subtracted from MOLA. The resultant value is multiplied by the atomic weight of A. This calculation returns the number of grams of A which were not involved in the reaction. Line 190 prints the result. In line 200 the number of moles of the product which were produced are calculated. RTAC, the coefficient specified for A in relation to C, is divided by RTAB. This value is multiplied by RTBA. The next computation divides RTCA, C's coefficient, by the result. Finally, MOLB is multiplied by the resultant value, and the value is assigned to MOLC. Control then branches to line 270 as a result of line 210.

If A is the limiting reagent, lines 220 through 260 are executed. Line 220 contains the identifying REM statement, and line 230 prints the message. When line 240 is executed, the amount of B which is unreacted is calculated. First the ratio of B to A is determined. MOLA is then multiplied by the ratio. This computation yields the number of moles of B which were consumed. To determine the moles of B which remain, this value is subtracted from MOLB. Since AMTU is to contain the number of grams which were not used, the number of unused moles is muliplied by the atomic weight of B. The resultant value is printed by line 250. Line 260 calculates the moles of C which were produced.

Lines 270 through 340 comprise the closing routine. Line 270 is a REM statement. In line 280, the grams of C which were produced, GRMC, are calculated. Lines 290 and 300 print the values of MOLC and GRMC. When line 310 executes, the person running the program is asked whether the program should be executed again. Line 320 evaluates the input value. If the value is not "Y", lines 330 and 340 end the program.

Ideal Gas Law

The ideal gas law consist of an equation with one constant and 4 variables. It states that for an ideal gas, the pressure multiplied by the volume is equal to the universal gas constant, the number of moles, and the temperature multiplied together. The value of any of the variables can be determined when the values of the other variables are known. The following program uses a menu and subroutines to solve for the unknown variable:

```
Ok
LIST
10 REM***IDEAL GAS LAW***
20 CLS: PRINT TAB(18) "MENU"
30 PRINT "1: N = number of moles"
```

program continued on next page

```
40 PRINT "2: P = pressure"
50 PRINT "3: T = temperature"
60 PRINT "4: V = volume"
70 PRINT: INPUT "Enter the unknown variable";UNKNOWN
80 IF UNKNOWN = 1 THEN GOSUB 1000
90 IF UNKNOWN = 2 THEN GOSUB 1100
100 IF UNKNOWN = 3 THEN GOSUB 1200
110 IF UNKNOWN = 4 THEN GOSUB 1300
120 IF UNKNOWN = 1 OR UNKNOWN = 2 OR UNKNOWN = 3 OR
    UNKNOWN = 4 THEN GOTO 150
130 PRINT "Invalid selection."
140 GOTO 90
150 INPUT "Do you wish to continue(Y/N)";ANS$
160 IF ANS$ = "Y" THEN GOTO 20
170 CLS
180 END
1000 REM*N UNKNOWN SUBROUTINE*
1010 CLS: INPUT "Enter pressure";P
1020 INPUT "Enter temperature in K";T
1030 INPUT "Enter volume in liters";V
1040 N = (P * V) / (.08206 * T)
1050 PRINT "The number of moles is";N
1060 RETURN
1100 REM*P UNKNOWN SUBROUTINE*
1110 CLS: INPUT "Enter number of moles";N
1120 INPUT "Enter temperature in K";T
1130 INPUT "Enter volume in liters";V
1140 P = N * .08206 * T / V
1150 PRINT "The pressure is";P
1160 RETURN
1200 REM*T UNKNOWN SUBROUTINE*
1210 CLS: INPUT "Enter number of moles";N
1220 INPUT "Enter pressure";P
1230 INPUT "Enter volume in liters";V
1240 T = P * V / (N * .08206)
1250 PRINT "The temperature is";T
1260 RETURN
1300 REM*V UNKNOWN SUBROUTINE*
1310 CLS: INPUT "Enter number of moles";N
1320 INPUT "Enter pressure";P
1330 INPUT "Enter temperature in K";T
1340 V = N * .08206 * T / P
1350 PRINT "The volume is";V
1360 RETURN
Ok
RUN
```

Lines 10 through 70 comprise the introduction and menu portions of the progam. The REM statement in line 10 identifies the program. Lines 20 through 60 print the menu. In line 70 the selection is requested and is stored in UNKNOWN.

Lines 80 through 110 branch to the appropriate subroutine. If UNKNOWN is equal to 1, the number of moles is to be determined. Line 80 then branches to the subroutine which begins in line 1000. A value of 2 indicates that the pressure is unknown, so line 90 calls the subroutine in lines 1100 through 1160. Line 100 calls line 1200 if UNKNOWN stores a value of 3. If unknown is equal to 4, the program branches to line 1300 as a result of line 110.

The comparisons in line 120 indicate whether or not a valid option was specified for UNKNOWN. If UNKNOWN contains a valid value, the program branches to line 150. Lines 130 and 140 execute if an invalid selection was made. The program then branches to line 90. Lines 150 through 180 comprise the ending routine.

The first subroutine begins in line 1000. Line 1000 identifies the subroutine with a REM statement. When lines 1010 through 1030 execute, the pressure, temperature, and volume are requested. Line 1040 calculates the number of moles, N. Remember that the ideal gas law states the following:

```
pressure * volume = .08206 * number of moles * temperature
```

Using this equation, the value for N can be found by multiplying .08206 by the temperature and then dividing the result into the left-hand side of the equation. This occurs in line 1040. In lines 1050 and 1060, the resultant value for N is printed, and the subroutine is exited.

The subroutines which compute the values of the pressure, volume, and temperature are similar in structure to lines 1000 through 1060. The value for P, the pressure, is computed by the subroutine which begins in line 1100. The values for the other variables are input when lines 1110, 1120, and 1130 execute. P is then computed in line 1140 by dividing the right-hand side of the equation by V, the volume. After this value is output in line 1150, the subroutine is exited.

Lines 1200 through 1260 execute if T, the temperature, is the unknown. The value for T is computed by dividing the left-hand side of the equation by .08206 multiplied by N. The result is then output, and the subroutine is exited.

The final subroutine, lines 1300 through 1360, only executes if V is the unknown. V's value is determined in line 1340 by dividing the right-hand side of the equation by P.

Physics

Since physics is basically a mathematical science, the same techniques which have been discussed in both this lesson and the last lesson can be applied to its study. In this section of the lesson, we will explore two examples of how to program the PC*jr* to solve physics problems.

Motion

Part of the study of elementary physics is the study of motion. Translational motion, or uniform motion in a straight line, is a concept which is introduced early and analyzed quantitatively. The PC*jr* can solve a translational motion problem if it is provided with a program such as the following:

```
Ok
LIST
10 REM***MOTION PROBLEMS***
20 REM*INPUT DATA
30 CLS
40 VS = 0:XS = 0:TS = 0:AS = 0
50 GOSUB 1000
60 K$ = K1$
70 GOSUB 1500
80 GOSUB 2500
90 K$ = K2$
100 GOSUB 1500
110 GOSUB 2500
120 REM*CALCULATIONS
130 IF VS AND XS THEN GOSUB 3000
140 IF VS AND TS THEN GOSUB 3500
150 IF VS AND AS THEN GOSUB 4000
160 IF XS AND TS THEN GOSUB 4500
170 IF XS AND AS THEN GOSUB 5000
180 IF TS AND AS THEN GOSUB 5500
190 REM*OUTPUT RESULTS
200 CLS
210 PRINT "V =";V
220 PRINT "X =";X
230 PRINT "T =";T
240 PRINT "A =";A
250 PRINT: PRINT "Do you wish to continue(Y/N)";ANS$
260 IF ANS$ = "Y" THEN GOTO 20
270 CLS
280 END
1000 REM*MENU SUBROUTINE*
1010 PRINT "V = velocity"
1020 PRINT "X = position"
1030 PRINT "T = time"
1040 PRINT "A = acceleration"
1050 PRINT: INPUT "Which two have known values"; K1$,K2$
1060 RETURN
1500 REM*ENTER VALUE SUBROUTINE*
1510 PRINT "For ";K$;", enter the value";
1520 INPUT VALUE
1530 RETURN
2500 REM*ASSIGNMENT SUBROUTINE*
2510 IF K$ = "V" THEN V = VALUE:VS = -1
2520 IF K$ = "X" THEN X = VALUE:XS = -1
2530 IF K$ = "T" THEN T = VALUE:TS = -1
2540 IF K$ = "A" THEN A = VALUE:AS = -1
```

program continued on next page

```
2550 RETURN
3000 REM*V,X KNOWN SUBROUTINE*
3010 T = (2 * X) / V
3020 A = V / T
3030 RETURN
3500 REM*V,T KNOWN SUBROUTINE*
3510 X = .5 * V * T
3520 A = V / T
3530 RETURN
4000 REM*V,A KNOWN SUBROUTINE*
4010 T = V / A
4020 X = V ^ 2 / (2 * A)
4030 RETURN
4500 REM*X,T KNOWN SUBROUTINE*
4510 V = 2 * (X / T)
4520 A = V / T
4530 RETURN
5000 REM*X,A KNOWN SUBROUTINE*
5010 V = (2 * A * X) ^ .5
5020 T = V / A
5030 RETURN
5500 REM*T,A KNOWN SUBROUTINE*
5510 X = .5 * A * T ^ 2
5520 V = A * T
5530 RETURN
Ok
RUN
```

Lines 10 and 20 contain REM statements which identify the program and the section, respectively. Line 30 clears the screen. When line 40 executes, the variables, VS, XS, TS, and AS, are initialized. These variables are used by the program to indicate which other variables have specified values.

The GOSUB statement in line 50 branches to the subroutine which operates the menu. Lines 1000 through 1060 comprise this subroutine. When lines 1010 through 1040 execute, the menu options are printed. Line 1050 identifies which two values are known. These values are stored in K1$ and K2$. The subroutine is exited in line 1060.

In line 60 the value in K1$ is assigned to K$. This process enables the subroutine in lines 1500 through 1530 to be used to input the specific values for both K1$ and K2$. Line 70 calls this subroutine, and the program branches to line 1500. After the subroutine is identified in line 1500, a prompt is printed as a result of line 1510. Line 1520 then requests the known value and assigns that number to VALUE. Line 1530 branches to the main program.

Line 80 contains another subroutine call. This subroutine is the assignment subroutine which is contained in lines 2500 through 2550. The REM statement in line 2500 identifies the subroutine. Lines 2510 through 2540 are responsible for assigning the value which is stored in VALUE to the appropriate variable. If K$ is equal to "V", then VALUE is assigned to V by line 2510. In addition, the variable VS is assigned a true value. VS indicates that a value for V was specified. If K$ contains a value of "X", then VALUE is assigned to X, and -1 is assigned to XS. This occurs in line 2520. Similar processes occur in lines 2530 and 2540 if K$ is equal to "T" or "A", respectively. Line 2550 branches back to the main program.

Lines 90 through 110 repeat the process for K2$. The value of K2$ is assigned to K$ in line 90. When line 100 is executed, the subroutine in line 1500 is invoked. Line 110 calls the assignment subroutine.

The REM statement in line 120 identifies this section of the program. Lines 130 through 180 call subroutines which calculate the values for the other two variables. Which subroutine is invoked is dependent upon the contents of VS, XS, TS, and AS. If VS and XS contain values of true, the program branches in line 130 to line 3000. If values for V and T have been specified, lines 3500 through 3530 will be executed after line 140. Lines 150 checks the status of VS and AS. Values of true in those variables will elicit a branch to the subroutine which begins in line 4000. Similarly, the values which are stored in XS and TS determine whether or not the program will branch to line 4500. The status of these variables is checked in line 160. Lines 170 and 180 contain similar subroutine calls which depend on the status of AS in conjunction with XS and TS respectively.

Line 190 contains a REM statement which identifies the next section of the program. In line 200 the screen is cleared. Lines 210 through 240 print the values contained in V, X, T, and A. When line 250 executes, the person who is running the program is asked whether or not the program should be executed again. If the answer is "Y", line 260 branches to line 20. Otherwise lines 270 and 280 clear the screen and end the program.

Line 3000 begins the subroutines which calculate the missing values. These calculations are based upon the following equations:

```
X = .5 * A * T ^ 2        Equation 1
V = A * T                 Equation 2
X = .5 * V * T            Equation 3
V = (2 * A * X) ^ .5      Equation 4
```

Notice that each equation involves three variables. Two variables must, by definition, have known values, but the value for the other variable is unknown. Through algebraic manipulation of these equations, the value for any of these variables can be determined. The subroutine in lines 3000 through 3030 assumes that the values for V and X are known. The REM statement in line 3000 identifies this point. T is computed by solving equation 3 for T. This occurs in line 3010. Line 3020 then calculates A based upon equation 2.

Lines 3500 through 3530 compute the values for X and A when V and T are known. Line 3510 uses equation 3 to calculate the value for X. That value is required by line 3520 to determine the value for A. When line 3530 is executed, control shifts to the main program.

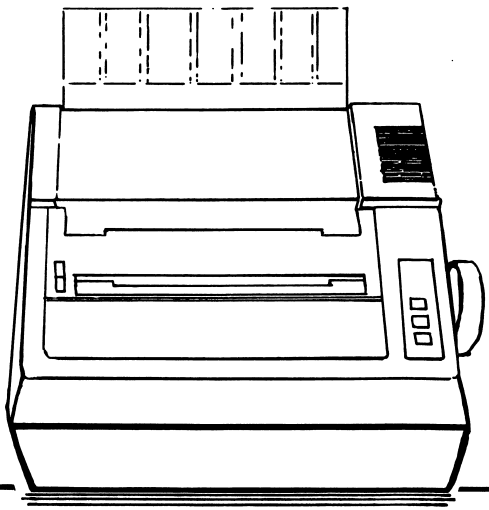
The remaining four subroutines parallel these. The equations are manipulated in order to compute the values of the unknowns. The main program is then called.

Work

A physics program does not need to be long to be effective. Some equations are used so often that you may find writing a simple program to solve such an equation to be very efficient. Determining how much work has been done, W, is a frequent problem. One formula which can be used to determine W is $W = FS \cos\theta$. The following program computes W's value using this formula:

```
Ok
LIST
10 REM***WORK PROBLEMS***
20 CLS: INPUT "Enter the force";F
30 INPUT "Enter the distance";DIST
40 INPUT "Enter the angle";ANG
50 INPUT "Enter the measure type";TYPE$
60 IF TYPE$ = "D" THEN ANG = ANG * .017453
70 W = F * DIST * COS(ANG)
80 PRINT "The work done =";W
90 PRINT: INPUT "Do you wish to continue(Y/N)";ANS$
100 IF ANS$ = "Y" THEN GOTO 20
110 CLS
120 END
Ok
RUN
```

Line 10 contains a REM statement which identifies the program. Lines 20 through 50 request data. If the angle measure is input in degrees, line 60 converts the measure to its radian equivalent. In line 70 W is calculated by multiplying together the force, the distance, and the cosine of the angle. Line 80 prints the resultant value. When line 90 executes, the person running the program is asked whether the program should be executed again. If the answer is "Y", line 100 branches to line 20. Any other answer causes lines 110 and 120 to be executed. In those lines the screen is cleared and the program ends.



Writing Papers and Reports

lesson 26

Lesson Goals

- *Learn how to use the word processing program HomeWord, created by Sierra On-Line, Inc., to write papers and reports*

Introduction

Papers, reports, and other written items can be created on the PC*jr* with the assistance of a word processing program and a printer. With a word processing program, papers can be easily modified and corrected without spending long hours at a typewriter. Word processing programs also allow multiple copies of a written item to be output, quickly and with little effort.

One of the more commonly used word processing packages for the IBM PC*jr* is HomeWord, by Sierra On-Line, Inc. One reason for HomeWord's popularity is that most people find it easy to learn.

Although almost any printer can be used with HomeWord to output standard text, a graphics printer is required if superscripts and subscripts are to be printed. A graphics printer is a printer which has the capacity to output graphics characters. The IBM-80 CPS Graphics Printer is an example. Since footnotes, which require superscripts, are used in the examples in this lesson, we recommend that you use a graphics printer. If such a printer is not available, disregard the superscripts in the exercise.

The discussion in this lesson will be in the form of a tutorial. We will be presenting in this tutorial the various steps which are involved in creating a typical report using HomeWord. It is a good idea to enter the commands as they are described. It is also a good idea to read the manual which accompanies HomeWord, since in this lesson we will give an overview of the program but will focus upon several techniques which are especially applicable to the writing of papers and reports.

Overview of HomeWord

The purpose of HomeWord is to create documents. A **document** is a combination of text and commands. With HomeWord, a document can be twelve pages in length if it is single spaced, longer if double or triple spacing is used. By storing pieces of a larger creation as individual documents, longer documents can be constructed.

HomeWord defines the screen into three areas. The largest of these areas is the **typing area**. The typing area comprises the upper two-thirds of the screen. The actual text of the document is entered in the typing area. A flashing cursor in this portion of the screen indicates that you can type the text.

The icons are contained in the left-hand side of the remaining area. This is the **menu area** of the screen. Any prompts will also appear here. When the menu is active, the cursor in the typing area does not flash, and a box will surround one of the icons. The box is the menu area's cursor.

The final screen area is a small box in the right hand corner of the screen. This box is the **outline area**. In the outline area the layout of the text is displayed. When the flashing cursor is present in the typing area, a miniature version of the cursor also flashes in the outline area. This smaller cursor indicates where on the page you are typing. If the menu is active, the outline area is replaced by two icons.

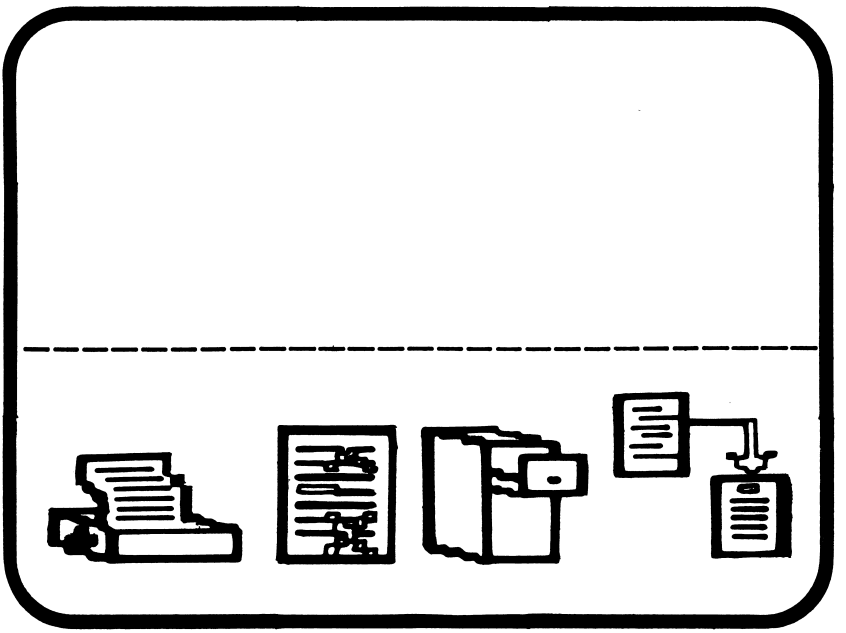


Figure 26.1. HomeWord's screen

HomeWord is a menu-driven program. The actual commands can be accessed through the selection of icons or in most cases by pressing a control key combination. The keyboard overlay which is provided with HomeWord specifies the key combinations. By using the control key, most commands can be immediately accessed.

In contrast, a series of icons may need to be selected before a command can be accessed. HomeWord has 43 icons. Between three and six icons are displayed on the screen. Only the lowest level icons directly access commands. These icons are displayed as a result of the selection of higher levels of icons. The right and left arrow keys are used to move the cursor, and pressing the Enter key selects an icon. Table 26.1 lists the icons which have control key equivalents, while Table 26.2 lists those commands which are only accessible through the menu.

Table 26.1. Commands which have control key equivalents

Level 3 Icons	Level 2 Icons	Level 1 Icons	Key Equivalents
Print	See final document		Ctrl-V
	Print document		Ctrl-P
	Starting page number		Alt-N
Edit	Copy text		Ctrl-C
	Erase text		Ctrl-E
	Move text		Ctrl-Y
	Find		Ctrl-F
	Find and replace		Ctrl-R
File	Get document		Ctrl-G
	Save document		Ctrl-S
	Erase document		Ctrl-X
	Insert document		Ctrl-A
Layout	Alignment	Align right	Ctrl-Z
		Align left	Ctrl-L
		Justify text	Ctrl-J
		Center next line	Ctrl-O
	New page		Ctrl-D
	Set spacing	Set top/bottom margins	Ctrl-Q
		Set left/right margins	Ctrl-K
		Set line spacing	Ctrl-W
		Set tab stops	Ctrl-T
	Print style	Boldface text	Ctrl-B
		Normal text	Ctrl-N
		Underline text	Ctrl-U
	Headings/footings		Alt-H

Table 26.2. Commands which lack control key equivalents

Level 3 Icons	Level 2 Icons	Level 1 Icons
Preset Values	Make backup documents?	
	Change preset margins	Change top/bottom margins Change left/right margins Change line spacing Change tab stops
	Save preset values	
	40/80 column screen	
	Type of printer	
Exit to DOS		

Starting HomeWord

In the following sections, we will create a paper with the assistance of HomeWord. In order to use HomeWord, however, the program must first be loaded. If the DOS is active, as indicated by the presence of the A> prompt, place the HomeWord diskette in the disk drive, type HW, and press the Enter key. If the prompt is not present perform the following steps:

Step 1. Place the HomeWord diskette in the disk drive.

Step 2. Boot the PC*jr*.

Step 3. The date and time are requested. If you wish, enter the date and time using the process which was described in lesson 5. Otherwise, press the Enter key twice.

When prompted to do so, remove the HomeWord diskette. Insert the data diskette on which the documents are to be stored. Press the Enter key.

Creating a Title Page

The PC*jr* is now waiting for your instructions. The cursor is flashing in an empty typing area. The empty typing area indicates a blank document. Since we want to create a new paper, we can simply begin to type at this point.

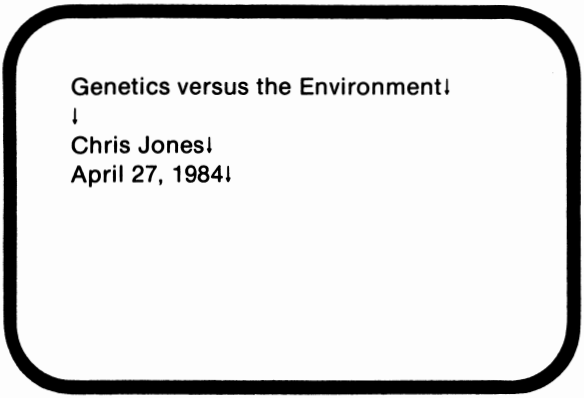
The first step in writing this paper is to create a title page. A title page usually includes at least the title of the paper, the author's name, and the date. Often this information is centered on the page and located one third of the way down the page. In the following sections we will explore the series of commands with which we can create the title page.

Enter Title Page Information

First we will enter the title page information. This step is simple. Just type the following lines:

Genetics versus the Environment↵
↵
Chris Jones↵
April 27, 1984↵

Once these lines have been entered, the typing area will appear as follows:



The downward pointing arrow which appears at the end of each line indicates that the Enter key was pressed. The arrow represents a carriage return. If the Enter key had not been pressed, the text on the screen would be treated as one line.

Notice that the second line only contains an arrow. This arrow indicates that a blank line should be inserted in the text. This blank line is included primarily for aesthetic purposes. Manipulating text for aesthetic purposes is known as **formatting**.

Change Top Margin

Earlier in this discussion we stated that we wanted to locate the title page information one third of the way down the page. This can be accomplished by either pressing the Enter key several times or by redefining the top margin so that the area on the page for text begins farther down. By default, the top margin consists of six blank lines. On the screen a page is defined as 66 lines of text at least two of which must be blank. By setting the top margin to 22, the text will begin one third of the way down the page. To redefine the top margin, enter the following sequence of commands:

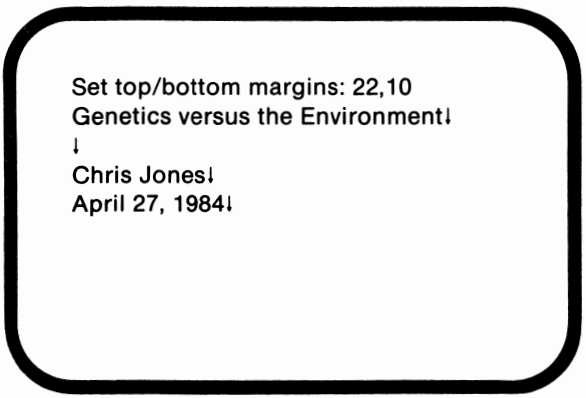
Step 1. Access the Set top/bottom margins command either through the menu or by pressing Ctrl-Q. This command will let us define a new value for the top margin.

Step 2. Next a prompt will appear in the menu area of the screen. This prompt indicates that the cursor should be moved to the spot in the text at which you want the margins redefined. Using the arrow keys, move the cursor to the G in Genetics and press the Enter key. This action indicates that we want all subsequent text to be relocated on the page.

Step 3. When the number of lines in the top margin is requested, the number 6 will appear under the flashing cursor. Replace the 6 with 22. Enter the value. The top margin now contains 22 blank lines instead of 6. Text will appear beginning at the 23rd line.

Step 4. When the bottom margin's value is requested, press the Enter key. The default value for the bottom margin, 10 lines, is still acceptable.

The typing area of the screen should have the following appearance:



Notice that the screen's top line contains the margin definition. Although this line appears on the screen, it will not appear when the document is printed.

A glance at the outline area indicates that the margins have been redefined. The lines which appear in this area have been moved down one third of the screen.

Define Line Spacing

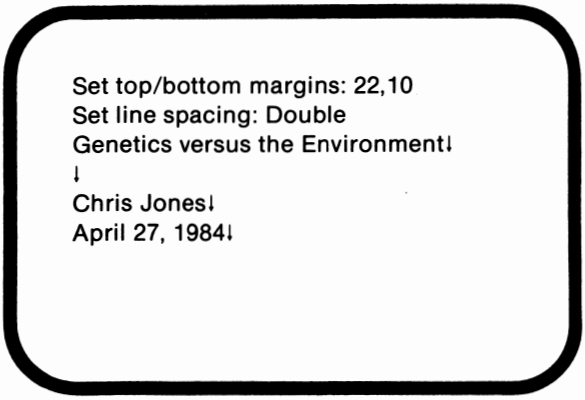
The next step is to select the line spacing. Most papers, including the one which we are creating, should be double spaced. By default, however, HomeWord selects single spacing. We can change this option by entering the following commands:

Step 1. Access the Set line spacing command. This command sets the line spacing.

Step 2. The cursor should be flashing over the G in Genetics. If not, relocate the cursor using the arrow keys. When the cursor covers the G, press Enter. All subsequent text will be double spaced.

Step 3. The prompt which appears in the menu area indicates that the line spacing default is single. Line spacing has three possible values: single, double, and triple. These values can be viewed by pressing the upward and downward pointing arrow keys. When the desired option, double, is encountered, press the Enter key. Pressing the Enter key indicates that the value has been selected. The line spacing is now double.

Notice the addition of the line spacing command to the screen:



A glance at the outline area indicates that the line spacing has indeed been changed. The lines throughout this document will be double spaced, unless a new option is selected.

Center Lines

Earlier we stated that the title page information is usually centered on the page. Centering the lines is the final step in creating a title page. HomeWord has a command which will automatically center text. The command, however, only applies to the next line of text. In order to format all of the title page information, the command must be entered three times. The following sequence of entries will accomplish this task:

Step 1. Access the Center next line command. This command does the centering.

Step 2. The cursor should be flashing over the G in Genetics. When it is, press Enter. This action centers the title.

Step 3. The next line to be centered is the author's name. Repeat Step 1 to access the command. Then move the cursor to the first letter in the author's name, since this is the next line to be centered. Press the Enter key.

Step 4. Only the date still must be centered. Repeat Step 1 and move the cursor to the A in April. Press Enter. The date is now centered.

Notice the addition of the three Center next line commands to the screen:

```
Set top/bottom margins: 22,10
Set line spacing: Double
Center next line
Genetics versus the Environment↓
↓
Center next line
Chris Jones↓
Center next line
April 27, 1984↓
```

The title page information has been centered, as a survey of the outline area indicates. The blank line was not centered because the command would have had no effect and thus was not necessary.

Main Text of the Paper

Now that the title page is finished, we can begin to work on the body, or main text, of the paper. This is generally the next step in the process of creating a paper. A number of actions are involved in creating the main text. In the following sections we will explore these actions.

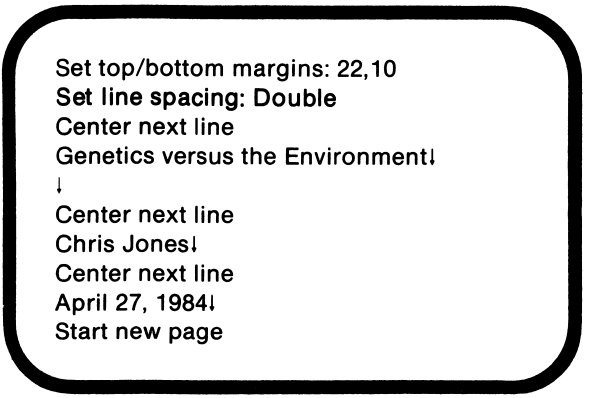
Start a New Page

If we were to start typing on the line after the date, our text would appear on the title page. Of course, we want the text to begin on a new page. HomeWord's Start new page command places all subsequent text on a new page. Enter the following series of commands:

Step 1. Access the Start new page command.

Step 2. The new page should start with the first line after the title page information. Move the cursor beneath the A in April. Press Enter to indicate that the new page should begin after the title page. Any text which we enter below the Start new page command will appear on a subsequent page.

The screen should now have the following appearance:



The outline area is now empty except for the cursor. The effect of most commands is evidenced in the outline area, and the Start new page command is no exception.

Enter Main Text

We can now begin to enter the text. In this section we won't worry about formatting. Instead, we will simply type. Type the information in figure 26.2.

Whether human behavior is controlled by our environment or at the whim of our genes is a matter of debate. At stake is the entire field of psychology, for if the genetic content of an individual is the determining factor in that individual's behavior, then psychology is merely a mislabeled form of biology, and clinical psychology is a fraud.↵

Human behavior can be described as a continuum; the upper and lower limits are established by genotype, and the exact point of expression is determined by the environment. The question of whether human behavior is genetically determined can then be translated into how broad is the continuum. If it is narrow, genetics is more influential; a broader spectrum emphasizes the environment. The size of the continuum, or the measure of the diversity in the expression of a trait, varies per behavior. Overall, however, I maintain that the environment is the stronger factor.↵

The existence of a behavior in both animals and humans or among all human societies is not proof of genetic determination. The genetic component may be very large. Culture and environment are pervasive influences, however, and it is seldom possible to study human behavior apart from the environment. Evidence indicates that by age two children have already assimilated a number of society's values such as sex role ideals. The same may be true for altruism and aggression. Values can be transmitted and are, unintentionally as well as intentionally. Environmental influences are difficult to alter but are still more variable than genetic heritage. Since much of human behavior changes over time, the proper emphasis in the nature-nuture debate should, in my opinion, be upon environmental factors. That alterability is indicative of the power and influence of the environment.↵

Figure 26.2. Sample paper

Although the text has now been entered, a number of formatting changes are still required. The text still begins one third of the way down the page, and the right edges are ragged. In addition, the first paragraph is a quotation. This quotation needs to be single spaced, indented, and footnoted. In the following sections, we will discuss these changes.

Redefine Margins

The first formatting change involves a redefinition of the margins. Currently the top margin contains 22 blank lines. Since we want the text to be located immediately below a one inch top margin, we need to return the top margin to its default value of 6 blank lines. A one inch top margin, or 6 lines, is standard for papers. The following combination of commands will return the margins to their default value:

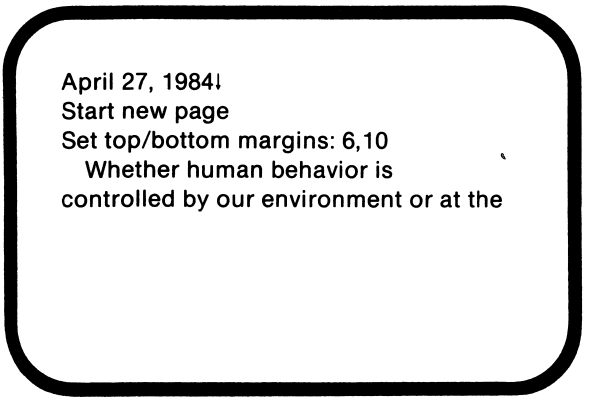
Step 1. Access the Set top/bottom margins command.

Step 2. Move the cursor using the arrow keys until it is located below the S in Start. Press Enter. The margins will be redefined after the completion of the title page.

Step 3. A cursor flashes over the value, 6, in the menu area. Press the Enter key to select this value.

Step 4. Press Enter again so that the bottom margin remains at 10 blank lines.

The top margin is now redefined to 6 lines. Glancing at the outline indicates this change. The screen should include these lines:



Unless we redefine the top/bottom margins again, all subsequent text will be located in lines 7 through 56 on the pages. The next problem is to align the right edges.

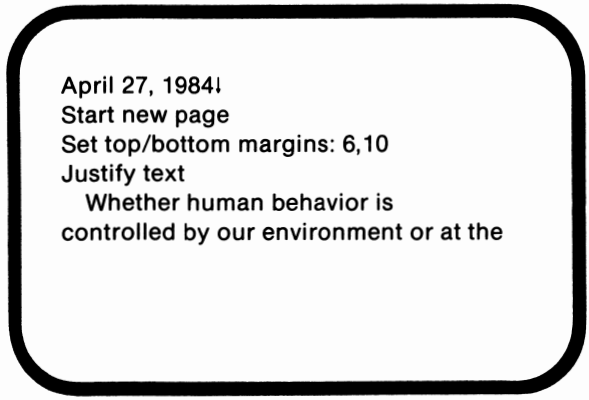
Justify Text

The formatting of the text would be improved by aligning the right edge of the text. Be default the text's left edge is aligned. This is the next step. Although the Align right command might seem to be the correct selection, we need to use the Justify text command. Text is justified when both the left and right margins are even. The Align right command will result in the right edge being even, but the left edge will be made uneven. To justify the text, use the following commands:

Step 1. Access the Justify text option.

Step 2. Locate the cursor under the S in Set, and press Enter. All subsequent text, unless otherwise directed, will be justified.

The Justify text command should be added to the screen, and the screen should contain the following lines:



The text is now justified, as a glance at the outline area demonstrates. HomeWord justifies text by padding the lines with spaces. In other words, more than 1 blank space will separate words in some lines so that all lines have the same length.

Indenting the Quote

As we stated earlier, the first paragraph is a quote. Quotations in a paper or report are usually indented. The next step, therefore, is to change the left and right margins so that the paragraph is indented. At the end of the paragraph, the margins must be redefined so that the text of the paper will not be similarly indented. The following series of commands will accomplish the first goal:

Step 1. Access the Set left/right margins command.

Step 2. The prompt requests that you move the cursor to the location at which you want the redefinition to begin. Move the cursor until it flashes under the J in Justify and press Enter.

Step 3. The left margin currently contains 10 spaces. Since it is standard for a quotation to be indented five additional spaces on each side, type in 15 when the number of spaces in the left margin is requested and press Enter.

Step 4. The number of spaces in the right margin is requested next. Again type 15 and press the Enter key.

At this point, all of the text has been indented. The series of commands which follows will reset the left and right margins to ten spaces beginning with the second paragraph.

Step 1. Access the Set left/right margins command.

Step 2. Move the cursor to the start of the second paragraph and press the Enter key.

Step 3. Press Enter to select 10 as the value for the left margin.

Step 4. Press Enter again to select 10 as the value for the right margin.

The screen should now include the following lines:

```
Justify text
Set left/right margins: 15,15
   Whether human behavior is
controlled by our environment or at the
whim of our genes is a matter of
debate. At stake is the entire field
of psychology, for if the genetic
content of an individual is the
determining factor in that individual's
behavior, then psychology is merely a
mislabeled form of biology and clinical
psychology is a fraud.↓
Set left/right margins: 10,10
```

In the outline area it is evident that only the first paragraph is now indented. This step is complete.

Single Space Quote

Another standard feature when dealing with quotes is that they are single spaced, instead of double. As such, our next step will involve changing the line spacing to single before paragraph 1 and returning it to double at the end of the paragraph. The following commands will set the line spacing to single:

Step 1. Access the Set line spacing command.

Step 2. Move the cursor so that it flashes below the S in Set left/right margins: 15,15. In other words, move the cursor to the beginning of the first paragraph. Press the Enter key.

Step 3. The line spacing prompt indicates that single is the default choice. Press Enter to select the single line spacing option.

The entire paper is now single spaced. Since we want only the first paragraph to be single spaced, we need to change the line spacing again with the following series of commands:

Step 1. Access the Set line spacing command.

Step 2. Move the cursor to the beginning of the line which includes the words, "Human behavior can be described as." Press Enter.

Step 3. The line spacing needs to be double. Use the arrows to view the choices, and press the Enter key when double appears in the menu area.

The typing area of the screen should now contain the following lines:

```
Justify text
Set left/right margins: 15,15
Set line spacing: Single
   Whether human behavior is
controlled by our environment or at the
whim of our genes is a matter of
debate. At stake is the entire field
of psychology, for if the genetic
content of an individual is the
determining factor in that individual's
behavior, then psychology is merely a
mislabeled form of biology and clinical
psychology is a fraud.↓
Set left/right margins: 10,10
Set line spacing: Double
   Human behavior can be described as
```

The quotation is now formatted properly, as a glance at the outline area indicates. The only step remaining with the quote is to footnote it.

Footnote the Quote

In a paper, quotes need to be footnoted. A footnote identifies the person being quoted and the location in which that quote can be found. A superscripted number is usually placed at the end of the quote. The specific number is determined by the number of footnotes preceding the quote. If the quote is the sixth footnote, it is assigned the number 6. The actual content of the footnote is then placed either at the bottom of the page or at the end of the paper.

To create a superscript with HomeWord, as we intend to do, printer control characters will be needed. The following commands will accomplish this goal:

Step 1. Locate the cursor on the arrow in the last line of the first paragraph.

Step 2. Press Fn-9. HomeWord defines Fn-9 as the Insert Character command. We need to insert the ASCII character which represents the ESC key, since ESC is part of the printer control character which creates superscripts.

Step 3. A prompt requests the ASCII decimal value which we want to insert. That value is 27, the value for the ESC key. Type 27 and press the Enter key. An arrow which points left is placed in the text. The last line of the paragraph should include the following characters:

```
psychology is a fraud.←↓
```

Step 4. Next type S0. ESC-S0 is a printer code which indicates that the following character or characters should be treated as a superscript. Do not insert any blank spaces.

Step 5. This is the first footnote, so type 1.

Step 6. Press Fn-9 again and repeat step 3. Another left arrow should appear.

Step 7. Type T. ESC-T indicates that the superscript is complete. The line should now appear as follows:

```
psychology is fraud.←S01←T↓
```

The main text of the paper is formatted. We still, however, have a number of processes to accomplish before the paper will be completed.

Footnotes

As we discussed earlier, footnotes may be placed at either the end of the paper or at the bottom of each page. In the following sections we will discuss both options.

Placing Footnotes at the End of the Paper

Placing footnotes at the end of the paper is simply a matter of typing the footnotes on a new page, giving that page a title, and structuring the page. In this section we will discuss the various steps which are involved.

Our first task is to create a new page for the footnotes. Footnotes which are placed at the end of a paper are located on a separate page. Enter the following:

Step 1. Move the cursor to the end of the document.

Step 2. Access the Start new page command and press Enter. This creates a page for the footnotes.

The next step is to change the line spacing to single so that the footnote will be properly formatted. The following commands will change the line spacing:

Step 1. Access the Set line spacing command.

Step 2. Move the cursor so that it is located under the S in Start. Press Enter.

Step 3. We want single spacing, so press Enter again when the prompt appears.

Now that the line spacing has been set to single, the next step is to enter the title. Type the following line:

Footnotes↵

The title currently appears at the upper left edge of the page. The title should, however, be placed in the center of the first line after the top margin. To center the title, do the following:

Step 1. Access the Center next line command.

Step 2. Move the cursor so that it covers the F in Footnotes. Press Enter.

Although the title has been centered, it also needs to be underlined. The sequence of commands which follows will underline the title:

Step 1. Access the Underline text command.

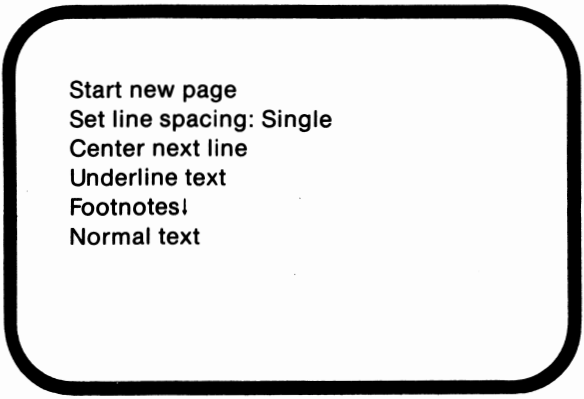
Step 2. Place the cursor over the F in Footnotes and press the Enter key.

The title is now underlined. If, however, we do not return the text to its normal status, all subsequent text will also be underlined. To correct this situation, perform the following steps:

Step 1. Access the Normal text command.

Step 2. Move the cursor so that it is located underneath the F in Footnotes. Press the Enter key.

The screen should include the following lines:



The title and the footnote page have been formatted. The next steps involve the actual creation of a footnote. To begin, perform the following steps:

Step 1. Press Enter once. This step is necessary because we have already changed the line spacing to single. We want the title to be somewhat apart from the actual footnotes.

Step 2. Press Tab once. A footnote is indented the same way a paragraph is.

In the next step we will create a superscript which identifies the footnote. To create the superscript, perform the following steps:

Step 1. Make sure the cursor is flashing at the first tab stop of an otherwise blank line.

Step 2. Press Fn-9.

Step 3. Type 27 and press Enter.

Step 4. Type S01.

Step 5. Repeat steps two and three.

Step 6. Type T.

Notice that this process is identical to the one with which we created the superscript for the quote. Our next step is to type the actual footnote. After typing the following line, end it with a blank space:

J. E. Williams,

In this step the author's name was entered. In a footnote the title of the book follows the author's name and is underlined. Since the next entry includes the title, enter the following sequence of commands:

Step 1. Access the Underline text command.

Step 2. Press the Enter key to insert the command.

Step 3. Type the following without pressing the Enter key:

Controversies in Psychology

Step 4. Access the Normal text command and press the Enter key.

Step 5. Type the remainder of the footnote:

(New York: Psybooks, 1984), p. 193.↵

The footnote page is now complete. The screen should include the following lines:

```
Start new page
Set line spacing: Single
Center next line
Underline text
Footnotes↓
↓
  ←S01←TJ. E. Williams,
Underline text
Controversies in Psychology
Normal text
(New York: Psybooks, 1984), p. 193.↓
```

If other footnotes had been used in the paper, they would follow the first footnote, be listed in order of occurrence, and be separated from preceding footnotes by one blank line.

Placing the Footnotes at the Bottom of the Page

Next we will discuss the procedure for locating footnotes at the bottom of the page on which the quote or data appears. This is a comparatively complicated process. Usually, placing the footnotes at the end of the paper will be sufficient, but occasionally interspersing footnotes throughout the text is required. This method involves inserting the note at the proper location and structuring the note and the surrounding text.

The first step is to examine the document and locate the first word in the next-to-last line on the first page of text. That word is altruism. The last three lines of text appear as follows:

> values, such as sex role ideals. The same may be true for altruism and aggression. Values can be transmitted and are, unintentionally as well as intentionally.

Next, we need to move the last two lines to the next page by performing the following steps:

Step 1. Move the cursor to the space which precedes the word altruism.

Step 2. Access the Start new page command. Press the Enter key.

Moving these two lines to the next page creates room for the footnote. Since footnotes are single-spaced, the next step is to change the line spacing to single. Enter the following commands:

Step 1. Access the Set line spacing command.

Step 2. Move the cursor to the S in Start new page. Press Enter.

Step 3. We want single spacing, so press Enter again.

We are now ready to begin typing the footnote. After pressing the Tab key once, enter the following series of commands to create the superscript:

Step 1. Press Fn-9.

Step 2. Type the value 27 and press the Enter key.

Step 3. Type S01.

Step 4. Repeat steps 1 and 2.

Step 5. Type T.

After creating the superscript, we can next type the author's name. Type the following and insert a blank space at the end:

J. E. Williams,

The title of the book is entered next. The title needs to be underlined. Perform the following steps:

Step 1. Access the Underline text command.

Step 2. Press the Enter key.

Step 3. Type the following without pressing the Enter key:

Controversies in Psychology

Step 4. Access the Normal text command and press Enter to reinstate normal text.

The next step involves typing other information about the source. Type the following remainder of the footnote:

(New York: Psybooks, 1984), p. 193.↵

Only one step remains. A glance at the outline area indicates that we should place additional blank lines between the footnote and the text. To accomplish this, enter the following series of commands:

Step 1. Move the cursor to below the S in Set line spacing.

Step 2. Press the Enter key twice to insert two additional blank lines. This action, however, also results in the last line being no longer justified. To ease the problem, insert an additional space between ideals and The.

This footnote is now complete. The screen should include the following lines:

```
values such as sex role ideals. The same may be true for ↓
Set line spacing: Single
↓
   ←S01←TJ. E. Williams,
Underline text
Controversies in Psychology
Normal text
(New York: Psybooks, 1984), p. 193↓
Start new page
```

At this point the paper is nearly finished. All that remains is to create a bibliography page and to number the pages. These will be discussed in the following sections.

Bibliography Page

The bibliography page is the page on which the sources which were used to write the paper are listed. Creating a page for the bibliography is very similar to creating a footnote page, although the actual structure of the page is a little different. In this section we will discuss these differences as we create the bibliography page.

A bibliography page is a new page. Thus the first step is to create a new page. Enter the following commands:

Step 1. Access the Start new page command.

Step 2. Locate the cursor at the end of the document. This can be accomplished very simply by pressing Fn-End. When the cursor is located at the end of the document, press the Enter key.

Now that a page for the bibliography has been created, the next step is to set the line spacing to single. If a footnote page was created, the line spacing should already be single. If not, enter the following commands:

Step 1. Access the Set line spacing command.

Step 2. Locate the cursor under the S in the Start new page command. Press Enter.

The next steps involve giving the page a title which is centered and underlined. First enter the following commands to center the title:

Step 1. Access the Center next line command.

Step 2. Move the cursor so that it is flashing beneath the last line of text on the screen. Press Enter.

The next step in this process, as we mentioned earlier, is to underline the title. Enter the following commands:

Step 1. Access the Underline text command.

Step 2. Locate the cursor under the C in Center. Press the Enter key.

At this point it is time to type the page's title. Type the following lines:

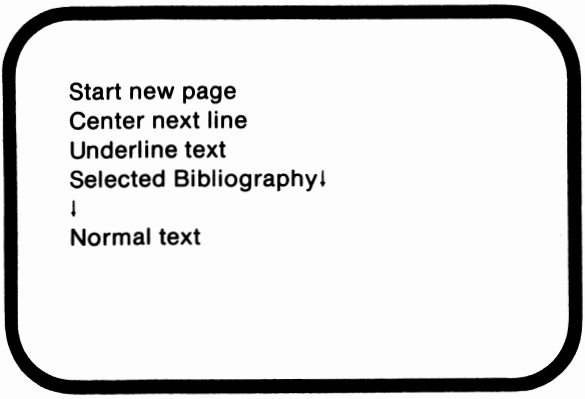
```
Selected Bibliography↵
↵
```

The page now is identified. The only step which remains involves switching the text back to normal so that only the title will be underlined. To do so, enter the following:

Step 1. Access the Normal text commands.

Step 2. When the cursor is located below the second downward arrow, press the Enter key.

At this point the screen should include the following lines:



In this paper we have only one source, or one bibliographic entry. A bibliographic entry has a different structure than does a footnote. Instead of indenting the first line, the second and any other

subsequent lines are indented. The commas in a footnote entry are replaced by periods in the bibliographic entry, and the name of the author is presented last name first. When multiple sources are used, they are listed alphabetically instead of in order of usage. These distinctions will become clearer after performing the various steps in this discussion.

Step 1. Enter the following line. Include two blank spaces after the second period:

```
Williams, J. E.
```

Step 2. The next part of the entry is the title of the book. As such, it needs to be underlined. Access the Underline text command and press the Enter key.

Step 3. Type the following line:

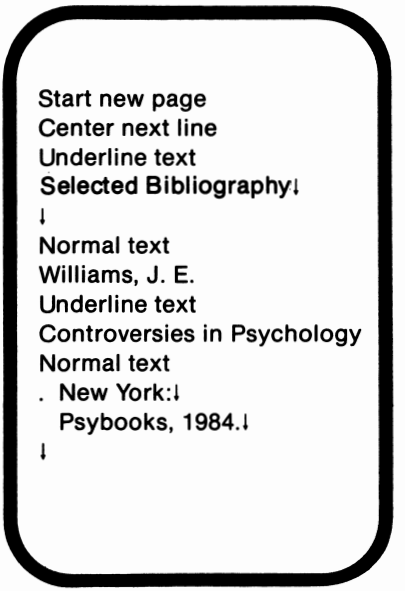
```
Controversies in Psychology
```

Step 4. Return the text to normal by accessing the Normal text command and pressing Enter.

Step 5. Type the following lines. Follow the period in the first line with two spaces. Tab the second line.

```
.    New York:↓
  Psybooks, 1984.↓
↓
```

The bibliography page is now complete. The screen should include the following lines:



Page Numbers

The final process in the creation of this paper is the numbering of the pages. HomeWord has special functions which will assist in this process. To number the pages, enter the following commands:

Step 1. Access the Headings/Footings command. This command creates headings or footings of thirty characters or less.

Step 2. Move the cursor to below the S in Set top/bottom margins: 6,10 on the first page of the main text. This means that the page number will appear on all pages except for the title page. Press the Enter key.

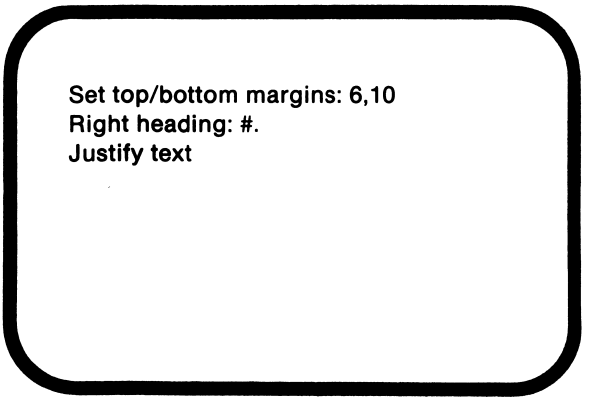
Step 3. A prompt indicates that the text should appear as a heading. Since the page number is usually placed in the upper right-hand corner of a paper, press Enter.

Step 4. A new prompt asks whether the text should appear on the right. Press Enter again.

Step 5. The text of the heading is requested. We want the text to include the page number followed by a period. A pound sign represents the page number since that number varies. Type the following:

#.↵

The heading has been inserted into the text, as a glance at the outline area will indicate. At this point the screen should include the following lines:



While the headings has been inserted, the page numbers have not. To begin the page numbering, enter the following commands:

Step 1. Access the Starting page number command.

Step 2. A prompt requests the page number on the document's first page. Enter 0 so that the first page of the main text will be numbered page 1.

The pages are now numbered, but no new commands are added to the screen. The Starting page number command is executed without being listed.

Appendix A.
ASCII Character Codes

In the following table, the ASCII codes will be given with any associated characters and control characters (for codes 0-31).

If you wish to display these characters, you can do so by issuing the following statement:

PRINT CHR$(*x*)

where *x* is the ASCII code of the character being displayed.

ASCII Value*	Character	Control Character	ASCII Value	Character	Control Character
000	(null)	NUL	016	▶	DLE
001	☺	SOH	017	◀	DC1
002	●	STX	018	↕	DC2
003	♥	ETX	019	!!	DC3
004	♦	EOT	020	π	DC4
005	♣	ENQ	021	§	NAK
006	♠	ACK	022	▬	SYN
007	(beep)	BEL	023	↨	ETB
008	◘	BS	024	↑	CAN
009	(tab)	HT	025	↓	EM
010	(line feed)	LF	026	→	SUB
011	(home)	VT	027	←	ESC
012	(form feed)	FF	028	(cursor right)	FS
013	(carriage return)	CR	029	(cursor left)	GS
014	♫	SO	030	(cursor up)	RS
015	☼	SI	031	(cursor down)	US

* Decimal

ASCII Value	Character	ASCII Value	Character	ASCII Value	Character
032	(space)	071	G	110	n
033	!	072	H	111	o
034	"	073	I	112	p
035	#	074	J	113	q
036	$	075	K	114	r
037	%	076	L	115	s
038	&	077	M	116	t
039	'	078	N	117	u
040	(	079	O	118	v
041	)	080	P	119	w
042	*	081	Q	120	x
043	+	082	R	121	y
044	,	083	S	122	z
045	-	084	T	123	{
046	.	085	U	124	|
047	/	086	V	125	}
048	0	087	W	126	~
049	1	088	X	127	⌂
050	2	089	Y	128	Ç
051	3	090	Z	129	ü
052	4	091	[	130	é
053	5	092	\	131	â
054	6	093	]	132	ä
055	7	094	^	133	à
056	8	095	_	134	å
057	9	096	`	135	ç
058	:	097	a	136	ê
059	;	098	b	137	ë
060	<	099	c	138	è
061	=	100	d	139	ï
062	>	101	e	140	î
063	?	102	f	141	ì
064	@	103	g	142	Ä
065	A	104	h	143	Å
066	B	105	i	144	É
067	C	106	j	145	Œ
068	D	107	k	146	Æ
069	E	108	l	147	ô
070	F	109	m	148	ö

ASCII Value	Character	ASCII Value	Character	ASCII Value	Character
149	ò	188	╝	227	π
150	û	189	╜	228	Σ
151	ù	190	╛	229	σ
152	ÿ	191	┐	230	µ
153	Ö	192	└	231	τ
154	Ü	193	┴	232	Φ
155	¢	194	┬	233	Θ
156	£	195	├	234	Ω
157	¥	196	─	235	δ
158	Pt	197	┼	236	∞
159	ƒ	198	╞	237	∅
160	á	199	╟	238	ϵ
161	í	200	╚	239	∩
162	ó	201	╔	240	≡
163	ú	202	╩	241	±
164	ñ	203	╦	242	≥
165	Ñ	204	╠	243	≤
166	ª	205	═	244	⌠
167	º	206	╬	245	⌡
168	¿	207	╧	246	÷
169	⌐	208	╨	247	≈
170	¬	209	╤	248	°
171	½	210	╥	249	∙
172	¼	211	╙	250	·
173	¡	212	╘	251	√
174	«	213	╒	252	n
175	»	214	╓	253	2
176	▒	215	╫	254	■
177	▒	216	╪	255	(blank 'FF')
178	▒	217	┘		
179	|	218	┌		
180	┤	219	■		
181	╡	220	▄		
182	╢	221	▌		
183	╖	222	▐		
184	╕	223	▀		
185	╣	224	α		
186	║	225	ß		
187	╗	226	Γ		

Appendix B.
PC*jr* BASIC Reserved Words

Reserved words are words which have a special meaning in BASIC. They include all BASIC commands, statements, function names, and operator names.

Reserved words are not allowed to be used as variable names in BASIC statements. Also, reserved words must be delimited in BASIC statements so that they can be recognized. Generally, words can be delimited through the use of blank spaces or special characters.

PC*jr* BASIC Reserved Words

ABS	DIM	IOCTL$	OR	SPACE$
AND	DRAW	KEY	OUT	SPC(
ASC	EDIT	KEY$	PAINT	SQR
ATN	ELSE	KILL	PALETTE	STEP
AUTO	END	LEFT$	PCOPY	STICK
BEEP	ENVIRON	LEN	PEEK	STOP
BLOAD	ENVIRON$	LET	PEN	STR$
BSAVE	EOF	LINE	PLAY	STRIG
CALL	EQV	LIST	PMAP	STRING$
CDBL	ERASE	LLIST	POINT	SWAP
CHAIN	ERDEV	LOAD	POKE	SYSTEM
CHDIR	ERDEV$	LOC	POS	TAB(
CHR$	ERL	LOCATE	PRESET	TAN
CINT	ERR	LOF	PRINT	TERM
CIRCLE	ERROR	LOG	PRINT#	THEN
CLEAR	EXP	LPOS	PSET	TIME$
CLOSE	FIELD	LPRINT	PUT	TIMER
CLS	FILES	LSET	RANDOMIZE	TO
COLOR	FIX	MERGE	READ	TROFF
COM	FN*xxxxxxxx*	MID$	REM	TRON
COMMON	FOR	MKDIR	RENUM	USING
CONT	FRE	MKD$	RESET	USR
COS	GET	MKI$	RESTORE	VAL
CSNG	GOSUB	MKS$	RESUME	VARPTR
CSRLIN	GOTO	MOD	RETURN	VARPTR$
CVD	HEX$	MOTOR	RIGHT$	VIEW
CVI	IF	NAME	RMDIR	WAIT
CVS	IMP	NEW	RND	WEND
DATA	INKEY$	NEXT	RSET	WHILE
DATE$	INP	NOISE	RUN	WIDTH
DEF	INPUT	NOT	SAVE	WINDOW
DEFDBL	INPUT#	OCT$	SCREEN	WRITE
DEFINT	INPUT$	OFF	SGN	WRITE#
DEFSNG	INSTR	ON	SHELL	XOR
DEFSTR	INT	OPEN	SIN	
DELETE	IOCTL	OPTION	SOUND	

Appendix C.
Answer Key

Lesson 1

True or false: 1. F 2. T 3. F 4. T 5. F 6. F 7. T
Multiple choice: 1. C 2. D 3. D 4. B 5. A 6. D

Lesson 2

True or false: 1. F 2. F 3. T 4. T 5. F
Multiple choice: 1. E 2. A 3. C 4. D 5. B

Lesson 3

True or false: 1. F 2. T 3. F 4. F 5. F
Multiple choice: 1. C 2. D 3. E 4. D 5. A

Lesson 4

True or false: 1. F 2. T 3. T 4. F 5. F
Multiple choice: 1. E 2. C 3. D 4. E

Lesson 5

True or false: 1. F 2. F 3. F 4. T 5. F
Multiple choice: 1. B 2. A 3. D 4. E 5. A

Lesson 6

True or false: 1. F 2. T 3. F 4. T 5. F
Multiple choice: 1. C 2. D 3. D 4. C

Lesson 7

True or false: 1. F 2. F 3. T 4. F 5. T
Multiple choice: 1. C 2. C 3. A 4. E 5. D

Lesson 8

True or false: 1. T 2. F 3. T 4. F 5. T
Multiple choice: 1. C 2. B 3. C 4. E

Lesson 9

True or false: 1. F 2. F 3. T 4. T 5. T
Multiple choice: 1. B 2. E 3. E 4. B 5. C

Lesson 10

True or false: 1. T 2. F 3. T 4. F 5. F
Multiple choice: 1. E 2. D 3. D 4. E 5. C

Lesson 11

True or false: 1. F 2. F 3. T 4. F 5. T
Multiple choice: 1. D 2. D 3. B 4. D 5. E

Lesson 12

True or false: 1. F 2. T 3. F 4. T 5. T
Multiple choice: 1. C 2. C 3. D 4. A

Lesson 13

True or false: 1. F 2. F 3. T 4. T 5. F
Multiple choice: 1. B 2. C 3. A 4. D

Lesson 14

True or false: 1. F 2. T 3. T 4. T 5. F
Multiple choice: 1. E 2. A 3. B 4. B

Lesson 15

True or false: 1. F 2. T 3. T 4. F 5. F

Multiple choice: 1. B 2. A 3. C 4. D 5. A

Computer exercise:

```
 10 FOR A = 1 TO 5
 20 INPUT "Enter a month (1-12)";MON
 30 GOSUB 1000
 40 IF MON = 1 OR MON = 2 OR MON = 3 THEN PRINT "The month is
    in the winter.":GOTO 70
 50 IF MON = 4 OR MON = 5 OR MON = 6 THEN PRINT "The month is in
    the spring.":GOTO 70
 60 IF MON = 7 OR MON = 8 OR MON = 9 THEN PRINT "The month is in
    the summer." ELSE PRINT "The month is in the fall."
 70 NEXT A
 80 END
1000 REM*ERROR CHECK
1010 IF MON<1 OR MON>12 THEN INPUT "Try again";MON
1020 RETURN
```

Lesson 16

True or false: 1. F 2. F 3. T 4. F 5. F

Multiple choice: 1. B 2. A 3. B 4. D

Lesson 17

True or false: 1. F 2. F 3. F 4. F 5. T

Multiple choice: 1. C 2. A 3. D 4. D

Lesson 18

True or false: 1. T 2. T 3. T 4. F 5. T

Multiple choice: 1. A 2. B 3. A 4. B 5. B

Lesson 19

True or false: 1. T 2. F 3. F 4. T 5. F

Multiple choice: 1. C 2. C 3. B 4. B 5. D

Lesson 20

True or false: 1. T 2. F 3. F 4. F 5. T
Mulitple choice: 1. A 2. A 3. C 4. C 5. A

Lesson 21

True or false: 1. F 2. F 3. T 4. F 5. T
Mulitple choice: 1. C 2. C 3. D 4. D 5. A

Lesson 22

True or false: 1. T 2. F 3. T 4. F 5. F
Mulitple choice: 1. D 2. C 3. D 4. E

Lesson 23

True or false: 1. F 2. F 3. F 4. T
Mulitple choice: 1. D 2. E 3. B 4. E

Index

A

A
  GDL 417-418
  hexadecimal 232
  PLAY 438
Abacus 28-29
ABS 348-349
Absolute form 403
Absolute value 348
Active page 396
Addition 249-250, 467
Add-on device 59
Address 47
Adjusting screen 101
Aiken, Howard 33-34
Align left 516, 527
Align right 516, 527
Allen, Paul 84
Altair 84
Alternate mode 105
Alt key 105
Alt key combinations 106
  Alt-/ 255
  Alt-A 105
  Alt-Ctrl-→ 101
  Alt-Ctrl-← 101
  Alt-Ctrl-Del 102
Ampersand in string formatting 287-288
Analog computers 27-28
Analytical engine 30, 32
AND 263
Applications software 86-87
Arctangent 344
Area of a triangle 481-483
Argument 346
Arithmetic and logic 132
Arithmetic machine 28-30
Arithmetic operators 34, 249-259, 265
Arrays 327-330, 333-335, 338, 374
Articulation 440
ASC 363
ASCII codes 227, 229, 261, 363, 381-382
ASCII-text conversion 363
Aspect 409
Aspect ratio 411-412
Assignment statements 236-237
Asterisk in numeric formatting 284-285
ATN 343-345
Audio jack 433
AUTO 149, 154-158

B

B
  GDL 415
  hexadecimal 232
  LINE 407-408
  PLAY 438
Babbage, Charles 30
  analytical engine 30, 32

Background color 381
low resolution graphics mode 391
screen 1 393
screen 4 394
text mode 388-390
with PRESET 405-406
Background music 440
Background titles 423, 429
Backslash in string formatting 288-290
Backspace key 176, 187
Backup documents 517
Base
arrays 333
EXP 350
mathematics 251
Base conversion 425, 427
BASIC 83
history 84
Version C 95
Version J 95
BASIC prompt 101
BEEP 109, 434
BF 407
Bibliographic entries 540-542
Bibliography page 540-543
Binary 24-25, 28, 46, 48
Binary patterns 424-428
Bit 46-47, 68
Blank documents 518
Blank line 120, 519, 523, 539
Body of a paper 523-533
Boldface text 516
Boolean operators 262
Border color
text mode 388-390
low resolution graphics mode 391
Boundary color 420-421, 423
Boxes, with LINE 406-408
Branching 312-313, 315-316
Built-in functions 343
Bytes 46-47, 62

C

C
GDL 416-417, 420-421
hexadecimal 232
PLAY 438
Cable 52
Calling 343
Caps Lock key 102, 104
Caret in numeric formatting 286-287
Carriage return/line feed 119-120, 273-274, 518-519
Cartridge BASIC 85-86, 95-96, 398, 437
Cartridges 59, 75-76
CASI 213-215
Cassette adaptor cable 75, 209-210
Cassette BASIC 85, 95, 423, 437
Cassette player/recorder 59, 74-75, 209-216
Cassette tapes 74
CBASIC 84
CDBL 352-353
Center lines 522-523, 534, 540
Center next line 516, 522-523, 534, 540
Center pixel 391
high resolution graphics 395
low resolution graphics 391
medium resolution graphics 392
Center point 409
Change left/right margins 517
Change line spacing 517
Change tab stops 517
Change top/bottom margins 517
Characters 416
lowercase 102
uppercase 102
Chords 437
CHR$ 229, 363
CHR$ expressions 424
CINT 352-353
CIRCLE 86, 408-410
CLEAR 394, 397, 424
CLS 125, 141
COBOL 83
Coefficient
exponential notation 231
mathematics 479
Cold boot 101
Colon 315-316
Color 381, 423
COLOR 110, 389-390, 393-395
Color burst signal 396
Color monitor 54-55
Columbia University 31, 34
Columns 333, 387-388, 390-392, 394-395, 399
Commas
in numbers 230
in numeric formatting 281-282
with PRINT 272, 274

Communications 52, 73
cordless 52
parallel 68, 70
serial 68-69
Compatibility PCjr-PC 393, 433
Compiled code 84
Compiler 85
Composite video monitor 54, 55
Compound expressions 259-260
Computers 18
as tools 18-19
definition 23
functions 23-27
history 28-35
impact 18
literacy 18
types 27-28
Computer-Tabulating-Recording Company 31
Concatenation 357
Conditional control 26, 30
Conditional statements 309, 315-316
Connector ports 49
Connectors 43, 49
Constants 228, 230, 234
mathematics 479
Control function 26
Control key combinations
Alt-Ctrl-→ 101
Alt-Ctrl-← 101
Alt-Ctrl-Del 102
Ctrl-Fn-End 178, 194-195
Ctrl-Fn-Home 178, 195-196
Ctrl-PgDn 178, 196-197
Ctrl-PgUp 179, 197-198
Coordinate 387, 403
Coordinate-specific distances 485
Copy text 516
COS 343-345, 459-463
Cosecant 490-491
Cosine 344
Cotangent 491
CR/LF 273-274
CSNG 352-353
Ctrl Key 107
Current line 137
Cursor 100
HomeWord 514
Cursor Down key 175, 180-181
Cursor Left key 175, 183-184
Cursor location 374-375
Cursor movement editing 173, 201-203
Cursor Right key 175, 181-183
Cursor Up key 175, 179-180

D

D
exponents 231, 234
GDL 411
hexadecimal 232
PLAY 438
Daisy wheel 66, 67
Daisy wheel printer 66-67
Dash 152
Data 227
numeric 229-235
string 227-229, 235
DATA 336-338
Data diskette 518
Data processing 25-26
Date 97-98, 517-518
Decimal notation 232
Decimal point in numeric formatting 277-278
DEF 382-383
Default values 161
Delay loops 464
Delay routines 464-465, 483, 497-498
DELETE 149, 164-167
Delete key 176, 184-185
Del key 176, 184-185
Density 64
Digital adding machine 28-30
Digital computers 27-28
DIM 333-336
Directive cards 31
Directory 219
Discriminant 480-481
Disk controller board 51-52
DISKCOPY 217-219
Disk drive 59-65, 216
operation 64-65
Diskette drive 41, 51, 59
Diskette envelope 61
Diskettes 59-65, 219
erasing files 221-222
formatting 216-219
Disk Operating System 62
Disks 59, 61
Distance formula 484-485
Division 467

Division by zero 255
Document 514, 518
Document length 514
Document manipulation 516
Dollar sign in numeric formatting 282-284
DOS 62
DOS 2.1 62, 217
DOS commands
  DISKCOPY 217-219
  FORMAT 216-217
DOS diskette 95
DOS prompt 100, 216-217
Dot matrix printer 66-67, 69
Dotted notes 445-446
Double density 64
Double-precision 231, 232-235, 352-353
Double sided diskettes 64
DRAW 86, 410-423
DRAW commands 411-422
Dummy arguments 382-383
Duration
  counts 436
  delay loops 464
  note 433, 435-436
  sound 448-450

E

e 350-351
E
  exponents 231-233
  GDL 412
  hexadecimal 232
  PLAY 438
Eckert, J. Presper 34
Edit 173
  cursor movement 173, 201-203
  EDIT command 173, 199-201
  line entry 173-174
EDIT 173, 199-201
EDIT command editing 173
Editing keys 107, 174-198
Edit mode 178, 193
Efficiency 479
Electronic Numerical Integrator and Calculator 34-35
ELSE 310-311, 315-316
Empty string 302, 305
END 117, 123-124, 313
Endnotes 533-537
Enhanced model 41
ENIAC 34-35
Enter key 108
Entry formats 98
Entry model 41
Erase 396
ERASE 338
Erase document 516
Erase text 516
Escape key 177
ESC key 177, 188, 532
ESC-S0 532
ESC-T 532
Exclamation point in string formatting 290-291
Exclusive OR 264
Exit to DOS 517
EXP 350-351
Expansion slots 43, 49
Exponent 231, 251, 350
Exponential notation 231, 286-287
Exponentiation 251-252, 257
Exposed read/write head slot 61
Expressions 249
External speaker 433-434

F

F
  GDL 412
  hexadecimal 232
  PLAY 438
Filename extension 222
Filenames 222
FILES 219
Final document 516
Find 516
Find and replace 516
FIX 347-348
Fixed length string field 288-291
Fixed point numbers 230-231, 278
Flags 468, 471, 487-489
Flashing characters 381, 388, 390
Floating point division 252-253, 467
Floating point numbers 230-231
Floppy diskette drive 41
FN 235, 382-383
FN key 104
FN key combinations 105-107
  FN-Break 155, 178, 193-194, 303, 304, 313

FN-End 177, 191
FN-Home 177, 192
FN-Prt Sc 104
Function keys 104
Footings 516
Footnote page 533-537
Footnotes 513, 531-539, 541-542
FOR 316-318, 320-321, 330, 332-333, 464
Foreground color 381, 417
low resolution graphics 391, 404
screen 1 393, 404, 417
screen 2 404
screen 4 394, 404
screen 5 404
screen 6 404
text mode 388-390
Foreground music 440
FORMAT 216-217
Format string 274-275
Formatting a diskette 216-219
Formatting characters 275
numeric 275-287
string 287-292
Formatting text 519
FORTRAN 83
Forty/eighty column screen 517
FRE 373-374
numeric argument 373
string argument 373
Frequency
note 435-436
tonal voice 2 449
Function keys 104
Functions 343
calling 343
definition 382
format 343

G

G
GDL 412
PLAY 438
Gates, William 84
GDL 410-423
GET 86
Get document 516
GOSUB 313-314
GOTO 312-313
Graphics 387-429
definition 387
modes 388
Graphics Definition Language 410
Graphics printer 513
Green keys 104

H

H 412
Harvard Mark I 33-35
Harvard University 33
Headings 516, 543-544
Headings/footnotes 516, 543-544
Hexadecimal numbers 83, 230, 232
tiling 424-429
High resolution graphics 394
aspect ratio 409
colors 395
memory requirements 395
screen dimensions 394
text columns 395
Hollerith, Herman 31, 33
tabulating machine 31-33
Home 177
HomeWord 513-544
overview 514-517
screen 514-515
start-up procedure 517-518
HomeWord commands 515-517
Housekeeping 373-374
Hybrid computers 27-28
Hypotenuse 343-344

I

IBM 31, 33-34
IBM Automatic Sequence Controlled Calculation 33
IBM Color Monitor 54-55
IBM Compact Printer 69
IBM Graphics Printer 69, 70, 513
IBM PC 62, 388
IBM PCjr Attachable Joystick 70-71
IBM PC XT 62, 388
Icons 514-515
IF 309-311, 315-316, 471
Immediate mode 115, 133-134
Impact dot matrix printer 69

Indenting quotes 528-530
Index hole 61-63
Index variable 317-318, 320, 330
Infinite loop 318
Infrared optical link 52, 71, 91
Initializing variables 468-470, 487-488, 507-508
Input 23-25, 132
INPUT 301-302, 305, 328
INPUT$ 302-304
Insert character 532, 535, 538
Insert document 516
Insert key 176, 185-186
Insert mode 176, 185
Ins key 176, 185-186
INSTR 364-365
INT 346-348, 380
Integer division 254-255
Integers 230, 232, 235, 254, 278, 352-353
Intel Corporation 43, 44
Intel 4004 calculator chip 44
Intel 8008 microprocessor 44
Intel 8080 microprocessor 44
Intel 8085 microprocessor 44
Intel 8088 microprocessor 43, 44, 46, 47
Intel 8088/8086 microprocessor 44
Internal Modem 51, 59, 73-74
Internal speaker 433-434
International Business Machine Co. 31, 33-34
Interpreter 85

J

Jacquard, Joseph-Marie 30
  punched cards 30
Joysticks 59, 70, 376
Justify 528
Justify text 516, 527-528

K

Kemeny, John G. 84
Keyboard 41, 52-53, 102-108
Keyboard connection cable 52-59, 71-72
Keyboard entries 109
Keyboard overlay 515
KEY OFF 421-422
Keys 52
  programmable 52
KILL 221-222
Kurtz, Thomas E. 84

L

L
  GDL 411
  PLAY 438-439, 442
Lake, Clair D. 33
Last point referenced 388
LEFT$ 358-360
Left-justified 289
LEN 365-366
Length conversion 496, 498
LET 236-237
LINE 406-408
Line entry editing 173-174
LINE INPUT 304-305
Line numbers 116-117, 121
Line patterns 428
Line spacing 516-517, 521-522, 530-531, 533, 537-538, 540
LIST 126, 142, 149, 151-153
Literals 291-292
LOAD 215-216, 220-221
Loading programs
  from cassette 214-216
  from diskette 220-221
LOG 351-352
Logarithm 351
Logical complement 263
Logical operators 262-265
Looping 316
Low resolution graphics 390-392
  aspect 409
  colors 391
  screen dimensions 390-392
  text columns 391
LPR 388, 391, 403, 407, 414-416, 420-421

M

M 414
Magnetic data storage 60
Make backup documents 517
Mantissa 231
Margins 516-517, 519-521, 526-530
Mass conversion 496, 498
Mark I 33-35
Mauchly, John W. 34
MB 440
Medium resolution graphics 391-394
  aspect 409
  colors 392-394

memory requirements 392-394
screen dimensions 391
text columns 392
Memory 48, 132
locations 375-376
random access 48
read-only 48
screen requirements 392-395
unused bytes 373-374
Memory and Display Expansion Board 49, 50, 59, 72-73
Memory pointer 136
Menu 465
Menu area 514-515
Menu-driven programming 465-468, 481-484, 487-488, 495-497, 503-508, 515
Metric conversions 495-498
MF 440
Microprocessor 43-44, 46-47
8-bit 47
16-bit 47
Microsoft BASIC 84, 100
Microsoft Corporation 62, 84
MID$ 360-361
Mill 31
Minus sign in numeric formatting 279-280
Mixing variable types 261
ML 440-441
MN 440-441
Mnemonics 83
MOD 255-256
Modem 73
Modes 388
Module calls 458
Modules 457
Level 0 457, 459, 463
Level 1 457, 459-460, 463
Level 2 457, 460-461, 463
Level 3 461
Modulo arithmetic 255-256
Monitors 54-55
Move text 516
MS 440-441
Multiplication 257, 467
Music buffer 440
Music legato 440
Music normal 440
Music staccato 441

N

N
GDL 416
PLAY 441
Natural logarithm 351
Negation 249, 258
Negative sign 273, 277
Nested loops 320-321, 333
NEW 126, 143, 243
New page 516, 524, 533, 537, 540
New page command 516, 524, 533, 537, 540
NEXT 316-318, 320-321, 330, 332-333, 464
NOISE 448-450
Noise voice 434, 448
Normal text 516, 534, 536, 538, 541-542
NOT 263
Note length 438-439
Number cards 31
Numbers 230, 246, 273
Numeric comparisons 261
Numeric constants 230
Numeric data 229-235
Numeric formatting 275-287
Numeric-string conversion 361-362

O

O 442-443
Octal numbers 230, 232
Octaves 436, 442, 446-447
Operands 249
Operating system 86
Operation cards 31
Operators 249
arithmetic 34, 249-259, 265
Boolean 262
logical 249, 262-265
relational 249, 261-262, 265
OPTION BASE 335-336
OR 264
Order of evaluation 265-266
Order of operations 260-261
Outline area 514-515, 524
Output 26-27, 132
Overflow 350-351

P

P
  GDL 419-421
  PLAY 443-444, 446
Page 396
Page definition 519
Page numbers 516, 543-544
PAINT 86, 423-429
Paint color 420, 423
PALETTE 86, 393-394
Palette patterns 427-428
Palettes 392-420
  fixed 392
  flexible 393, 397
  screen 1 392-393
  screen 4 393-394
  screen 6 395
PALETTE USING 86
Papers 513-544
Parallel communications 68
Parallel Printer Attachment 70
Parameters 358
Parentheses 266
Pascal, Blaise 28
  arithmetic machine 28-30
Pascal language 83
Pauses 443-444, 446
PEEK 375-376
Pennsylvania, University of 34
Percent sign in numeric formatting 276
Periodic sounds 448-449
Peripherals 59
Permanent label 61
PI 409
Pitch of a note 433, 435
Pixels 387, 390-392, 394, 403, 423
PLAY 86, 437-448
Plus sign in numeric formatting 278-279
Pointers 338, 388
POKE 376
Ports 49
POS 374-375
Position 507-508, 510
Pound sign in numeric formatting 275-277
Power supply board 49-50
Power supply/transformer 41, 49, 53
Power switch 94
Precision 232
  double 232-234
  integers 232
  single 232-234
PRESET 404-406
Preset values 517
Pressure 503-504, 506
PRINT 83-84, 110, 115, 118-120, 122, 271-274, 293
Print document 516
Printer control characters 531-532
Printers 59, 66-70
  types 66
  use with HomeWord 513, 517
Print style 516, 534, 536, 538, 540-542
PRINT USING 271, 274-275
Print zones 272, 275
Problem solving 455
Program 23, 83, 86
Program lines 116-117
Programmable keys 52
Programmable tone generator 433-438, 440
Programming language 83
  assembly 83
  compiled 84-85
  high level 83
  interpreted 84-85
  machine 83
Program mode 115, 134
Program structure 459, 486
Prompt 301, 304, 514
PSET 404-406
Punched cards 30-31, 33-34
  directive cards 31
  number cards 31
  operation cards 31
PUT 86

Q

Quadratic equation 479-481
Quadratic formula 479-481
Quotation 526-533
Quotation marks 228-229

R

R 411
Radians 344, 459-463, 487, 490
Radius 409
RAM 43, 48, 59, 72, 376
Random 376-377
Random access memory 48
Random data access 60
RANDOMIZE 380
Random number generator 376-380
Random number sequences 377-380
READ 336-338
Read-only memory 48
Read/write head 59-60, 62
Reciprocal 491
Relative form 403-404
Relational operators 261-262, 265
REM 117, 123-124, 313, 462
Remington Rand Corp. 34
RENUM 149, 158-163
Reports 513-544
Reserved words 235, 337, 338
Rests 443
RETURN 313-314
RIGHT$ 359-360
Right-justified 277
RND 376-380
ROM 43, 48, 75
Rotational angle 417-418
Rows 333, 387-388, 390-392, 394, 399
RUN 115, 124-125, 135-140, 149-150
Run-time monitor 85

S

S 413
Sample paper 525
SAVE 212-214, 220, 222
  cassette 213-214
Save document 516
Save preset values 517
Saving programs
  cassette 212-214
  diskette 220, 222
Scale 413
Scaling 398
Scaling factor 413
Screen 272, 387
  0 387-390, 396
  1 391-393, 396, 427
  2 394, 396, 426
  3 390-392, 396, 428
  4 391-394, 396, 427
  5 391-392, 394, 396, 428
  6 394, 396, 427
  coordinates 387
  modes 388
SCREEN
  command 110, 395-397
  function 381-382
Screen width 272
  HomeWord 519
Searching 516
Secant 491
Sectors 62-63
Seed 378-380
See final document 516
Semicolons
  GDL 423
  PLAY 444
  PRINT 272-274
Semiconductor chip 44
Sequential data access 60, 74
Sequential execution 309
Serial Adapter Cable 69
Serial communications 68
Set left/right margins 516, 528-530
Set line spacing 516, 521-522, 530-531, 533, 537-538, 540
Set tab stops 516
Set top/bottom margins 516, 519-521, 526-527
Sierra On-Line, Inc. 513
SGN 349-350
Shift key 102
Sign 349
Significant variable names 468, 472
Simple expressions 259
SIN 343-345, 459-463
Sine 344
Single density 64
Single-precision 232-236, 352-353
Single sided diskettes 64
Soft sector method 62
Software 86
  classifications 86

SOUND
command 434, 445
statement 435-437
Sound effects 433
Sound generators 433
8253 timer 433
SN 76489A 433-438, 440
Sound numbers 448-449
Sound statements and commands:
BEEP 434
NOISE 448-450
PLAY 86, 437-448
SOUND 434-437, 445
Source 448-450
Source code 84
Source diskette 217
SPACE$ 292, 295-296
SPC 292-295
SQR 345-346
Square root 345-346
Starting page number 516, 544
STEP 317-318, 403, 414
Storage function 24
Storage unit 31
STR$ 361-362
STRING$ 366-367
String comparisons 262, 364
String concatenation 357
String constant 228-229
String data 227-229, 235
String formatting 287-296
String length 365-366
String-numeric conversion 361-362
String replacement 361
Strings 227-230, 273
String search 364-365
Style, with LINE 407
Subroutines 313-314, 483, 486-492, 503-510
Subscripted variables 327-336
Subscripts 328, 335, 513
Substrings 421
Subtraction 258-259, 467
Superscripts 513, 531-532, 535-536, 538
Swapping 218
System board 43, 45
System date 97-98
System reset 101-102, 107
System start-up 91-101
with disk drive 95-101
without disk drive 93-95
System time 98-99
System unit 41, 43-44

T

T 444
TA 418-419
TAB 292-295
Tab key 177
insert mode off 189
insert mode on 190
Tables 327, 331-335, 374
Tab stops 516-517
Tabulating machine 31-33
Tabulating Machine Company 31
TAN 343-345, 459-463
Tangent 344
Tape leader 213
Target diskette 217
Tempo 444
Temporary label 61
TERM 86
Termination condition 423, 429
Text-ASCII conversion 363
Text columns 388, 391-392, 395
Text layout 514-516, 527-528
Text manipulation 516
Text mode 388-390, 396
color burst signal 396
colors 388-390
dimensions 388
pages 396
THEN 309-311, 315-316, 471
Thermal dot matrix printer 69
Tile mask 426
Tiling 423-429
Time 98-99, 507-508, 510, 517
Timer 433
Title page 518-523
Tonal voices 436-438
Tone generator 433-438, 440
Top-down design 455-463
advantages 459
Top-down design chart 458-462, 468
Tracks 62-63
Trigonometric functions 343-345, 459-463, 486-492
Trigonometry 343
Tune definition language 437-448

Type identification characters 236
Typing area 514-515, 518

U

U 411
Underline text 516, 534, 536, 538, 540-542
Univac I 34
User-defined functions 373, 382-383
  definition 382-383
  names 382-383

V

V 445
VAL 361-362
Values 234, 236-237
Variable length string field 287-288
Variable names 234, 235, 472
Variables 234-243
  assigning values 236-237
  numeric 235
  processing 237-243
  string 235
  types 236, 261
  use with GDL commands 422-423
  use with PLAY 444
Variable storage 132, 238-239, 241, 338
Video display 54, 72
  devices 54
VIEW 86, 398-399
Viewport 398, 399
VIEW SCREEN 398-399
Visual page 396
Voices 435-438
Volume
  cassette recorder 211
  noise 448-450
  note 433, 435-436, 445
Volume conversions 497-498

W

Warm boot 101-102
Watson Computing Bureau 34
WEND 319-320, 471
WHILE 319-320, 471
White sounds 448-449
WIDTH 272, 375, 381
WINDOW 86, 398-399
Word processing 513-544
  advantages 513
Write protect notch 61

X

X
  GDL 421-422
  PLAY 447-448
XOR 264

Special Symbols

! (exclamation point)
  single-precision 233
  string formatting 290
(pound sign)
  double-precision 234
  numeric formatting 275
  page number 544
  PLAY 438
$ (dollar sign)
  numeric formatting 282
  string identification 236
% (percent sign)
  integer identification 236
  numeric formatting 276
  (caret) 251
& (ampersand) 287
* (asterisk)
  line number generation 155
  multiplication 257
  numeric formatting 284
- (minus sign)
  GDL 414
  negation 258
  negative sign 273
  numeric formatting 279
  PLAY 438
  subtraction 258
_ (underline)
  cursor 100
  underline 292
= (equal sign)
  assignment 236-237
  equals 262

+ (plus sign)
addition 249
concatenation 357
GDL 414
numeric formatting 278
PLAY 438
< (less than sign)
less than 262
lower octave 446
> (greater than sign)
greater than 262
raise octave 446
/ (slash) 252
. (period) 445
(backslash)
integer division 254
string formatting 288
↓ (cursor down)
CR/LF 519
cursor down 181
← (cursor left)
cursor left 184
ESC 532
→ (cursor right)
cursor right 182
memory pointer 136
↑ (cursor up) 180
↵ (enter) 100
÷ (division) 252
.BAS BASIC file 222
&H hexadecimal number 232
&O octal number 232
>= greater than or equal to 262
<> not equal to 262
<= less than or equal to 262
numeric formatting 286

$17.95

IBM PCjr® for Students

IBM PCjr for Students is designed to help students grades 7 through adult learn to use and program the IBM PC*jr*.

IBM PCjr for Students is divided into a series of 26 individual lessons. Each lesson features a hands on approach to learning. The student will actually be working at the computer as he progresses through each of the lessons.

The lessons are designed so that each one involves a discovery as well as a problem solving activity. Several lessons are dedicated to showing the student how the PC*jr* can be used to help him or her with school work. Interesting examples as well as exercises are included with each lesson.

ISBN: 0-938862-25-1 LC 84-50843